



Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software

Autor: Denise Vasques de Cerqueira
Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Brasília, DF
2013



Denise Vasques de Cerqueira

Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Brasília, DF

2013

Denise Vasques de Cerqueira

Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software/ Denise Vasques de Cerqueira. – Brasília, DF, 2013-

74 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Luiz Carlos Miyadaira Ribeiro Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. ROI. 2. Qualidade. I. Dr. Luiz Carlos Miyadaira Ribeiro Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software

CDU 02:141:005.6

Denise Vasques de Cerqueira

Aplicação de metodologia de análise de retorno sobre investimento no contexto do Centro de Qualidade e Testes de Software

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Trabalho aprovado. Brasília, DF, 11 de dezembro de 2013:

**Dr. Luiz Carlos Miyadaira Ribeiro
Júnior**
Orientador

Msc. Aline Lopes Timóteo
Convidado 1

Msc. Cristiane Soares Ramos
Convidado 2

Brasília, DF
2013

Este trabalho é dedicado aos meus pais, Isaurino e Benedita. Ao meu pai por ser um grande exemplo de que com esforço podemos chegar aonde desejamos e à minha mãe por me ensinar a ter fé.

Agradecimentos

A Deus pela oportunidade, pela sabedoria, pela paciência, pela força e pelo ânimo sempre renovado para passar por todas as etapas desse longo processo.

Aos meus pais pelo incentivo, pelo tempo disposto em me orientar, pelo investimento realizado na minha educação, pela confiança de que eu conseguiria, por todo amor, carinho, cuidado, compreensão e paciência a mim dedicados por toda a minha vida.

Aos meus irmãos e cunhados que mesmo em alguns momentos achando que escolhi um caminho longo e difícil a seguir profissionalmente, nunca deixaram de acreditar que eu seria capaz, me ajudaram e me incentivaram a todo o tempo, demonstrando que família é o que pode haver de mais importante.

Aos meus sobrinhos que me salvaram nos momentos de sufoco com seus lindos sorrisos e com seu amor incondicional, revivendo a doce criança que sempre vai existir dentro de mim.

Aos meus amigos e ao meu namorado que nunca saíram do meu lado, mesmo com o pouco tempo disponível que passei a ter para eles.

Aos meus amigos da faculdade por - apesar de terem sofrido junto comigo - terem me incentivado, me ajudado e depositado confiança em mim.

Aos meus mestres por todo o conhecimento repassado, conhecimento esse que seguirá comigo em todo o caminho profissional.

Sem vocês, nada disso teria sido possível.

*"Que os vossos esforços desafiem as impossibilidades,
lembrai-vos de que as grandes coisas do homem foram
conquistadas do que parecia impossível."
(Charles Chaplin)*

Resumo

Na busca pela qualidade, muitas empresas têm investido grandes somas em dinheiro nos seus processos de software, porém ao se decidir por realizar um investimento, devem ser considerados os benefícios potenciais e os custos financeiros advindos de tal decisão. A dúvida é se os benefícios, em contrapartida aos custos, que vêm dessas ações de tentativa de otimização dos processos, ultrapassam o investimento. O presente trabalho consiste em realizar um análise de retorno sobre investimento (ROI) em qualidade de software, de modo a analisar o impacto financeiro de práticas de melhoria de processo sob o ponto de vista de uma metodologia que considera a contagem de defeitos durante a atividade de detecção de defeitos e o esforço despendido nessas atividades, observando o impacto desse investimento em economia de custo e redução de cronograma. Tal metodologia foi selecionada pela decisão de realizar essa análise no contexto do Centro de Qualidade e Testes de Software (CQTS) da Universidade de Brasília, de modo que seja possível avaliar se o investimento em melhoria de processo e qualidade nos testes é vantajoso para o laboratório.

Palavras-chaves: ROI. qualidade. melhoria de processo. investimento. teste. retorno sobre investimento.

Abstract

In the search for quality, many companies have invested large amount of money in their software processes, however when deciding on making an investment, one must consider the potential benefits and financial costs arising from this decision. The question is if the benefits that come from these actions attempt to optimize the processes beyond the investment. The present work is to execute an analysis of return of investment (ROI) in software quality, in order to analyze the financial impact of process improvement practices from the point of view of a methodology that considers the defect count during the activity of detection of defects and effort spent on these activities, noting the impact of investment in cost reduction and schedule savings. This methodology was selected by the decision to perform this analysis in the context of the Centre for Software Quality and Testing (CQTS), at University of Brasilia, so that it is possible to assess whether the investment in process improvement and quality in testing is advantageous to the laboratory.

Key-words: ROI. quality. process improvement. investment. testing. return on investment.

Lista de ilustrações

Figura 1 – Processo de detecção de defeitos – Teste de Software.	32
Figura 2 – Processo de inspeção de software. Fonte: (KALLNOWSKI; SPINOLA, 2007).	32
Figura 3 – Processo de <i>debugging</i> . Fonte: (SOMMERVILLE, 2007).	32
Figura 4 – Processos de V&V e <i>debugging</i> acontecem de maneira intercalada. . . .	33
Figura 5 – Fluxo utilizado para aplicação da metodologia	48
Figura 6 – Processo de Gerenciamento de Demanda do CQTS.	50
Figura 7 – Processo de Testes do CQTS.	51
Figura 8 – Composição da planilha para registro dos resultados dos testes no CQTS. .	53
Figura 9 – Relatório de resultados dos testes automatizados por caso de teste. . .	53
Figura 10 – Relatório de resultados dos testes automatizados por suíte de teste. . .	53
Figura 11 – Modelo de registro do tempo gasto para execução dos testes manuais .	58
Figura 12 – Registro do tempo gasto para execução dos testes manuais	58
Figura 13 – Registro do tempo gasto para execução dos testes automatizados . . .	60
Figura 14 – Cálculo da média de esforço para construção de <i>scripts</i> com dados do Redmine	60
Figura 15 – Processo de Gerenciamento de Demanda do CQTS.	71
Figura 16 – Processo de Testes do CQTS.	73

Lista de tabelas

Tabela 1 – Modelo de qualidade do produto de software. Fonte: (ISO/IEC, 2010) .	27
Tabela 2 – Modelo de qualidade em uso. Fonte: (ISO/IEC, 2010)	28
Tabela 3 – Definição da escala de 5 fatores requerida para redução de prazo	43
Tabela 4 – Definição dos elementos de custo usados em um projeto de software. Fonte: (EMAM, 2005)	54
Tabela 5 – Definição dos elementos de custo associados a automação de testes . .	55
Tabela 6 – Definição dos elementos de custo associados a execução de testes manuais	55
Tabela 7 – Definição dos elementos para determinar o esforço de encontrar e corrigir defeitos. Fonte: (EMAM, 2005)	56
Tabela 8 – Adaptação dos elementos para determinar o esforço de encontrar e corrigir defeitos.	56
Tabela 9 – Resultado da execução dos testes manuais	58
Tabela 10 – Resultado da execução dos testes automatizados	59

Sumário

1	Introdução	21
1.1	Motivação	22
1.2	Objetivos	22
1.3	Hipóteses	23
1.4	Resultados Esperados	23
2	Referencial Teórico	25
2.1	Entendendo a qualidade de software	25
2.2	Medição de software	29
2.3	A importância de realizar verificação e validação	30
2.4	Automação de testes	33
2.5	Metodologias para análise de ROI em Qualidade de Software	35
2.6	Considerações Finais	44
3	Aplicação da Metodologia	47
3.1	Metodologia de Trabalho	47
3.2	O que é o Centro de Qualidade e Testes de Software?	49
3.3	Processos do CQTS	49
3.4	O CQTS e a automação de testes	50
3.5	Adaptação da metodologia	53
3.6	Execução de Testes	57
3.7	Aplicação do modelo para cálculo de ROI	60
3.7.1	O modelo financeiro de ROI	63
3.8	Interpretação dos Resultados	64
4	Considerações Finais	67
4.1	Sugestões de melhoria para o CQTS	68
4.2	Sugestões de trabalhos futuros	68
	Referências	69
ANEXO A	Primeiro Anexo	71
A.1	Descrição dos processos do CQTS	71
A.1.1	Gerenciamento de Demanda	71
A.1.2	Processo de Teste	72
A.1.2.1	Planejamento da Demanda	72
A.1.2.2	Execução de Testes	73

A.1.2.3	Construção de Testes	73
A.1.2.4	Automatização de Testes	74

1 Introdução

Ao se decidir realizar um investimento em algo, é necessário considerar o retorno financeiro que pode advir de tal investimento. Muitas companhias utilizam metodologias de cálculo de retorno sobre investimento (ROI) – que segundo [Rico \(2004\)](#) “[...] é a quantidade de dinheiro que retorna a partir de um investimento” – para basear suas escolhas em processos de tomada de decisão. Em tecnologia da informação não é diferente. Com o intuito de evitar a perda financeira que maus investimentos podem acarretar, as empresas devem considerar os custos e os benefícios dos investimentos feitos em seus processos de trabalho.

Assim como os investimentos em ferramentas, tecnologia e pessoal, os investimentos em qualidade também são uma decisão importante para o negócio, visto que os investimentos adequados em qualidade podem maximizar o lucro das organizações desenvolvedoras de software. De acordo com [Leal \(2002\)](#), “[...] ROI é o benefício que se obtém por cada unidade investida em tecnologia durante certo período de tempo e costuma utilizar-se para analisar a viabilidade de um projeto”.

Cabe salientar que um investimento em um projeto é viável quando o retorno financeiro é maior do que o investimento realizado para o desenvolvimento do projeto. E um investimento em qualidade de software quando bem executado, realizado com as ferramentas adequadas e por uma equipe capacitada, pode reduzir o custo do desenvolvimento de software.

Quando fala-se em ROI em qualidade de software, considera-se, além dos valores investidos no processo de garantia da qualidade como um todo e nas ações de melhoria que podem ser realizadas nesse processo, os valores investidos no processo de detecção de defeitos, que consideram os testes e as inspeções de software, sendo estes os momentos onde os defeitos serão detectados. É importante descobrir em que momento do processo de detecção de defeitos custa mais identificar tais defeitos e em que momento custa mais para corrigi-los. O ideal é que a detecção dos defeitos e a correção dos mesmos sejam realizadas antes que o software seja repassado ao usuário final, onde os custos com reparação de defeitos tornam-se muito mais altos.

Para que seja possível realizar uma análise de ROI em uma organização desenvolvedora de software no contexto de melhoria da qualidade do processo de testes, é necessário que a mesma possua um plano de detecção de defeitos que faça parte de um consistente processo de verificação e validação de software (V&V), onde, de acordo com [Bartié \(2004\)](#),

a verificação visa garantir o processo de engenharia de software e a validação está focada na garantia da qualidade do produto de software. Testar é a forma mais comum de se verificar a existência de problemas no produto final.

A metodologia a ser utilizada na aplicação da metodologia, a de [Emam \(2005\)](#), considera os defeitos encontrados durante o processo de detecção de defeitos para calcular o ROI de projetos pelas organizações.

Efetuar uma análise de ROI ajuda, também, a identificar quais categorias de custos associadas aos projetos são as maiores responsáveis por aumentar o custo do projeto.

1.1 Motivação

Constantemente, a alta direção das organizações desenvolvedoras de software se depara com a seguinte questão: O investimento que está para ser realizado vale ou não a pena? Quais as vantagens de se investir em uma tecnologia ou técnica e que retorno trará tal investimento?

Análises de ROI auxiliam no processo de tomada de decisão das organizações, pois permitem saber se o lucro obtido vale o investimento realizado, quais benefícios podem ser alcançados a longo prazo investindo-se em qualidade, além de permitir mapear quais custos impactam mais em um retorno financeiro baixo.

De acordo com [Emam \(2005\)](#),

A quantidade de dinheiro investido na melhoria da qualidade de software é uma decisão de negócios. Uma maneira de informar esta decisão é considerar como qualidade e lucro estão relacionados. Um argumento que pode ser feito é que o investimento adequado em termos de qualidade é o montante que irá maximizar o lucro.

Nesse contexto, é importante saber como utilizar o ROI no processo de tomada de decisão ao se investir em qualidade de software, de modo a prevenir perdas.

1.2 Objetivos

O objetivo deste trabalho é avaliar os ganhos obtidos em melhoria de processo de software, em um laboratório de pesquisa, a partir do uso de metodologia para a análise de retorno sobre investimento.

Para alcançar o objetivo geral deste trabalho, foram definidos os objetivos específicos abaixo:

- Realizar revisão bibliográfica sobre qualidade de software e assuntos correlacionados;

- Estudar metodologias para análise de retorno sobre investimento;
- Discutir e definir como avaliar melhoria de processo a partir da metodologia selecionada;
- Aplicar a metodologia; e
- Analisar os resultados obtidos.

1.3 Hipóteses

As suposições desta pesquisa são:

- Será possível determinar o retorno de investimento em automação de testes em um laboratório de pesquisa (CQTS) a partir do uso de metodologia para análise de ROI;
- A coleta dos dados para análise de ROI é economicamente viável para o laboratório de pesquisa.

1.4 Resultados Esperados

De acordo com os objetivos do projeto, como resultado da aplicação da metodologia, espera-se responder aos seguintes questionamentos:

- Considerando a realização dos passos propostos para a aplicação da metodologia, o que é mais vantajoso para a equipe do CQTS: manter o investimento em automação de testes ou manter o processo de testes considerando apenas testes manuais?
- Considerando as duas modalidades de testes, qual delas mostrou um ROI mais vantajoso para o CQTS?

Através das respostas aos questionamentos, a tomada de decisão sobre o investimento em qualidade de software terá como base a consideração dados concretos obtidos na fase de aplicação da metodologia.

2 Referencial Teórico

Esse referencial teórico relaciona os temas importantes para fundamentar a proposta deste trabalho. Encontra-se dividido em seções, de forma que cada uma aborda assuntos importantes para o entendimento do que tratará o projeto, conforme a seguir:

Na seção 2.1, são apresentados conceitos de qualidade de software. Esta conceituação é feita a partir da visão e da obra de diferentes autores, o que permite consolidar o conhecimento. É feita a distinção entre qualidade do processo e do produto, são apresentadas as visões de qualidade do produto e é mostrado um conjunto de características das visões de qualidade que podem ser medidas e avaliadas.

A seção 2.2 traz a conceituação da medição de software, também pela visão de diferentes autores, mostra qual a importância de se utilizar medição, fala sobre a diferença entre alguns conceitos chave da medição de software (medida, métrica e indicador), além de explicitar que mesmo que não seja possível alcançar um software livre de defeitos, é importante possuir um processo de garantia da qualidade. Nesta seção é listada uma série de benefícios que podem advir da implementação de um processo de garantia da qualidade dentro da organização.

Na seção 2.3, discute-se o quão importante pode ser possuir processos de V&V dentro de uma organização desenvolvedora de software, quais os benefícios de se testar e inspecionar durante o ciclo de vida de desenvolvimento e traz a diferenciação dos dois termos, com o intuito de definir a que objetivo se destina cada um.

A seção 2.4 apresenta uma breve descrição do que é a automação de testes, qual o intuito de seu surgimento e como pode auxiliar as equipes de teste com a redução de trabalho manual que proporciona, além de apresentar a diferença entre tipos de ferramentas de automação de testes.

A seção 2.5 apresenta as metodologias de diferentes autores utilizadas para análise de ROI em qualidade de software e, em mais detalhes, a metodologia eleita para ser aplicada no contexto do CQTS.

2.1 Entendendo a qualidade de software

Qualidade de software é a área da Engenharia de Software que visa garantir conformidade ao produto de software. De acordo com [Guerra e Colombo \(2009\)](#), a qualidade de software, “[...] objetiva garantir que especificações explícitas e necessidades implícitas

estejam presentes no produto, por meio da definição e normatização de processos de desenvolvimento”. Ainda de acordo com [Guerra e Colombo \(2009\)](#) “[...] as especificações explícitas nada mais são do que a definição dos requisitos, as condições e os objetivos propostos para o produto”. Segundo a [NBR ISO/IEC \(2001\)](#), “[...] as necessidades implícitas são necessidades reais que podem não ter sido documentadas”.

Há muitas definições do que é a qualidade de software e todas elas dependem do contexto a que se aplicam, das expectativas das partes envolvidas e dos resultados esperados da execução de projetos. Dessa forma, é possível afirmar que as visões sobre qualidade de software diferem entre os grupos envolvidos.

A qualidade de software pode ser dividida em dois conceitos chave: qualidade do processo e qualidade do produto. A qualidade do produto é apoiada pela qualidade do processo. Mesmo que os modelos aplicados durante a garantia da qualidade sejam aplicados ao processo, o objetivo é obter um produto que esteja alinhado às expectativas do cliente. A qualidade do produto depende da qualidade dos processos usados durante o desenvolvimento.

A [ISO/IEC \(2010\)](#) divide a qualidade do produto em qualidade do produto, propriamente dita, e qualidade em uso. De modo que para cada uma dessas visões, fornece também as características a serem avaliadas.

Para medir qualidade do produto, a [ISO/IEC \(2010\)](#) sugere um modelo de qualidade do produto de software composto por um conjunto de oito características de software que devem ser avaliadas e se relacionam com as propriedades estáticas de software e as propriedades dinâmicas do sistema de computador, sendo elas: Funcionalidade, Eficiência, Compatibilidade, Usabilidade, Confiabilidade, Segurança, Modularidade e Portabilidade. E dentro de cada uma dessas características, fornece um conjunto de subcaracterísticas a serem avaliadas, de modo a direcionar as avaliações de qualidade das organizações e prover melhoria da qualidade. Subcaracterísticas estas, que são descritas na Tab. (1).

Segundo a [ISO/IEC \(2010\)](#), a qualidade em uso de um sistema caracteriza o impacto que o produto tem sobre os *stakeholders*. A qualidade em uso é composta por um conjunto de 5 características com suas respectivas subcaracterísticas, conforme pode ser observado na Tab. (2).

[Bartíe \(2004\)](#) define que “[...] qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos”. Deve-se pensar em uma maneira de garantir a conformidade, nesse sentido, [Guerra e Colombo \(2009\)](#) definem que os requisitos devem estar definidos para “[...] permitir que sejam gerenciados com o uso de medidas, de forma a reduzir o retrabalho e aumentar a produtividade”.

Diante das definições acima expostas, pode-se visualizar a relação da qualidade de

Tabela 1 – Modelo de qualidade do produto de software. Fonte: (ISO/IEC, 2010)

Característica	Subcaracterísticas
Funcionalidade	<ul style="list-style-type: none"> - Completitude Funcional - Correção Funcional - Adequação Funcional
Eficiência	<ul style="list-style-type: none"> - Tempo comportamental - Utilização dos Recursos - Capacidade
Compatibilidade	<ul style="list-style-type: none"> - Coexistência - Interoperabilidade
Usabilidade	<ul style="list-style-type: none"> - Capacidade de aprendizado - Operabilidade - Proteção contra erros do usuário - Estética de interface de usuário - Acessibilidade
Confiabilidade	<ul style="list-style-type: none"> - Maturidade - Tolerância a falhas - Recuperabilidade
Segurança	<ul style="list-style-type: none"> - Confidencialidade - Integridade - Não-repúdio - Responsabilidade - Autenticidade
Modularidade	<ul style="list-style-type: none"> - Modularidade - Reusabilidade - Analisabilidade - Testabilidade
Portabilidade	<ul style="list-style-type: none"> - Adaptabilidade - Instabilidade - Substituição

software com a medição de software. Ambas são atividades que se complementam, uma vez que, para analisar a conformidade dos processos e produtos com as especificações explícitas e necessidades implícitas visando garanti-la, é necessário medir.

A partir das características e subcaracterísticas acima listadas, é possível derivar várias medidas que permitirão identificar onde há não conformidade.

Em organizações desenvolvedoras de sistemas, é cada dia mais importante minimizar os riscos de ocorrência de erros, e isso significa poupar a empresa dos custos provenientes desse fato. Segundo [Bartíé \(2004\)](#), “[...] mesmo sendo impossível atingir o tão esperado software *bug-free*, deve-se criar uma estrutura e processos que proporcionem um ambiente favorável e que permita a detecção do maior número possível de erros”. Assim, é possível visualizar o quão importante é para as organizações a existência de um processo de controle da qualidade.

Tabela 2 – Modelo de qualidade em uso. Fonte: (ISO/IEC, 2010)

Característica	Subcaracterísticas
Eficácia	- Eficácia
Eficiência	- Eficiência
Satisfação	- Utilidade - Confiabilidade - Prazer - Conforto
Liberdade de risco	- Mitigação econômica de risco - Mitigação de risco saudável e segura - Mitigação do risco ambiental
Cobertura de contexto	- Completitude do contexto - Flexibilidade

Por mais bem planejados e estruturados que sejam os processos de engenharia de software de uma empresa, eles não garantirão um software com “zero defeito”. Ainda de acordo com [Bartíe \(2004\)](#), “[...] devemos entender que em todas as aplicações de software, um conjunto de defeitos está incorporado à aplicação e pronto a se manifestar quando determinado conjunto de situações for combinado”.

E mesmo que os processos não garantam um software livre de defeito, a missão de descobrir defeitos cria a consciência da importância de um software com o mínimo de defeitos e cria atitudes relacionadas ao “zero defeito”, a não proliferação de defeitos, essa é uma percepção de que tudo que se faz pode comprometer a qualidade final do software.

[Bartíe \(2004\)](#) lista alguns benefícios para as organizações de se possuir um processo de garantia da qualidade, dentre os quais:

- **Tornar o ciclo de vida de desenvolvimento confiável** – relaciona-se com a minimização dos riscos de proliferação de erros por conta dos processos melhor idealizados, planejados e executados;
- **Garantir ações corretivas no ciclo de desenvolvimento** – relaciona-se com a identificação dos pontos do projeto que apresentam maior incidência de problemas. Tal identificação só é possível através de um levantamento estatístico referente aos erros identificados;
- **Evitar a ingerência do Projeto de Software** – relaciona-se com o fato de evitar que aconteça a perda da capacidade de gerenciar um projeto, que ocorre quando o gerente perde o controle do que está sendo desenvolvido, pelo fato de não existir um mecanismo que controle se o que está sendo construído tem a qualidade desejada;
- **Ampliar as chances de sucesso do projeto de software** – relaciona-se com a minimização dos diversos pontos críticos de um projeto de desenvolvimento de

software, pois agrega ao software confiabilidade, que é um fator fundamental para o sucesso de um projeto;

- **Ampliar a produtividade do desenvolvimento** – relaciona-se com a alocação de profissionais em atividades deficientes do projeto, o que vai reduzir a desorganização da equipe, refletindo na diminuição da quantidade de erros, na diminuição do retrabalho e no aumento da produtividade da equipe;
- **Evitar a propagação de erros** – relaciona-se com o objetivo de garantir que a manutenção ocorrida na aplicação não afete o comportamento de outras funcionalidades. Os testes devem se concentrar nas partes que sofreram as mais recentes modificações, evitando que as mudanças propaguem erros para outros módulos.

Segundo [Bartié \(2004\)](#), “[...] um bom processo de garantia da qualidade deverá potencializar os testes de verificação e validação de modo que os esforços sejam minimizados e os resultados sejam os positivos possíveis”.

2.2 Medição de software

A medição está diretamente ligada à quantificação de atributos ou conjunto de atributos, relacionada com as dimensões de qualidade. De acordo com [Sommerville \(2007\)](#), “[...] a medição de software se dedica a derivar um valor numérico para algum atributo de um produto de software ou de um processo de software”. A medição é utilizada para prover um melhor entendimento dos atributos dos produtos e para avaliar a qualidade dos produtos ou sistemas submetidos aos processos de desenvolvimento das organizações. E as medidas dos processos de medição são utilizadas para ajudar a construir software de alta qualidade.

[Pressman \(2006\)](#) afirma que a medição apoia a gestão, que, quando conduzida adequadamente, apoia o processo de tomada de decisão que pode conduzir o projeto ao sucesso. Ainda segundo [Pressman \(2006\)](#), um engenheiro de software precisa de critérios objetivos para ajudá-lo a dirigir o *design* do software (projeto de dados, arquitetura, interface e componentes) e as métricas de produto fornecem uma base a partir da qual é possível conduzir objetivamente e avaliar quantitativamente a análise, o projeto, a codificação e os testes.

De acordo com [Estácio e Valente \(2010\)](#), “[...] com o amadurecimento da engenharia de software a medição tem se tornado fundamental na coleta de dados significativos, na identificação de pontos positivos e falhos na organização, assim como no ciclo de vida de melhoria de processo”. Possuir um processo de medição ajuda a organização a gerenciar e basear sua tomada de decisão em fatos. Já de acordo com [DeMarco](#) (apud [ESTÁCIO; VALENTE, 2010](#)), “[...] não se pode gerenciar o que não se pode medir”.

Para entender o conceito de medição, é necessário que se tenha em mente a diferença entre os conceitos de métrica, medida e indicador. De acordo com [Pressman \(2006\)](#), uma **medida** é estabelecida quando um único ponto de dados for coletado, o número de erros descoberto em um único componente de software, por exemplo; uma **métrica** relaciona as medidas individuais de alguma forma, o número médio de erros encontrado por teste de unidade, por exemplo; e, segundo o Instituto de Engenharia Elétrica e Eletrônica (IEEE), um **indicador** chama a atenção para uma situação particular, normalmente está relacionado a uma métrica e provê a interpretação daquela métrica numa determinada situação ou contexto.

No que se refere ao software, indicadores fornecem profundidade na visão do processo, do projeto e do produto final para auxiliar no processo de tomada de decisão. Geralmente, a representação e comunicação são feitas por meio de tabelas ou gráficos. Segundo [Pressman \(2006\)](#), “[...] as métricas representam medidas indiretas, a qualidade em si nunca é medida, em vez disso, medimos indicadores de qualidade”.

Os processos de medição bem sucedidos procuram coletar e analisar dados para futuras tomadas de decisão, com o objetivo de nortear tal processo, possibilitando que a organização saiba o que fazer, quando fazer e onde fazer. Mas, para que a medição não se resuma apenas a coleta de dados, é necessário que esteja alinhada às necessidades da organização, possibilitando que se obtenha dados significativos.

2.3 A importância de realizar verificação e validação

Para saber se algo funciona corretamente e de acordo com as expectativas, o homem está habituado a testar tudo o que lhe é interessante usar, há a necessidade instintiva de saber se as coisas funcionam como deveriam. E com o desenvolvimento de software não é diferente. O ideal é que seja possível identificar faltas, falhas, erros ou defeitos antes que o produto seja entregue ao cliente e ao longo da construção do software. Após a sua implementação é necessário executar atividades que auxiliem a verificar se o produto corresponde às especificações e às necessidades do cliente, tais atividades são denominadas Verificação e Validação de Software (V&V).

De acordo com [Pressman \(2006\)](#), “[...] verificação se refere ao conjunto de atividades que garante que o software implementa corretamente uma função específica. Validação se refere a um conjunto de atividades diferente que garante que o software que foi construído corresponde aos requisitos do cliente”. Já [Sommerville \(2007\)](#) afirma que “[...] verificação envolve verificar se o software está de acordo com as especificações. Deve-se verificar se ele atende aos requisitos funcionais e não-funcionais especificados. [...] A finalidade da validação é assegurar que o sistema atenda às expectativas do cliente”.

A realização de testes é a forma mais comum de verificar se o produto atende às

especificações explícitas e às necessidades implícitas e se realiza aquilo que o cliente deseja. Porém, além dos testes de software existem as inspeções de software. Segundo [Sommerville \(2007\)](#), “[...] testes de software envolvem executar uma implementação do software com dados de teste. Examinam-se as saídas do software e seu comportamento operacional para verificar se seu desempenho está conforme necessário”. Já quanto as inspeções de software, [Sommerville \(2007\)](#) afirma que “[...] analisam e verificam representações de sistemas como documento de requisitos, diagramas de projeto e código-fonte de programa. Inspeções podem ser usadas em todos os estágios do processo.

[Sommerville \(2007\)](#) define que dois tipos de teste que podem ser usados em diferentes estágios do processo de desenvolvimento de software: os testes de validação, que têm a finalidade de mostrar que o software é o que o cliente deseja; e os testes de defeito, que se destinam a revelar defeitos no sistema em vez de simular o seu uso operacional. Portanto, não há uma delimitação rígida de definição a esses dois tipos de teste, uma vez que nos testes de validação é possível encontrar defeitos e durante os testes de defeito é possível ver que o sistema atende aos seus requisitos.

É importante ressaltar que o teste de software não é uma forma de garantir a qualidade do software e sim de atestar se a qualidade existe ou não. Segundo [Pressman \(2006\)](#), “[...] A qualidade é incorporada ao software durante o processo de engenharia de software. A aplicação adequada de métodos e ferramentas, revisões técnicas formais efetivas e gerência e medição sólidas, todas levam à qualidade que é confirmada durante o teste”. Um processo de teste de software pode ser observado na Fig. (1).

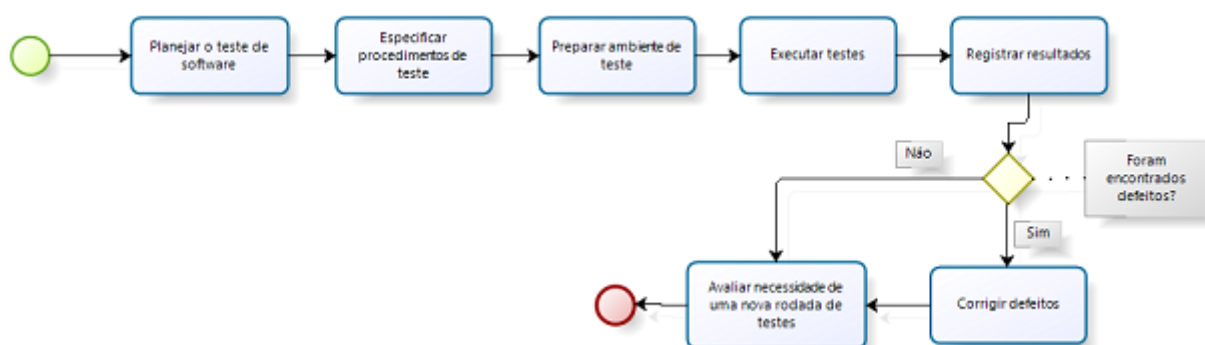


Figura 1 – Processo de detecção de defeitos – Teste de Software.

Segundo [Kallnowski e Spinola \(2007\)](#), “[...] a inspeção de software é um tipo particular de revisão que pode ser aplicado a todos os artefatos de software e possui um processo de detecção de defeitos rigoroso e bem definido”. Um processo de inspeção de software pode ser observado na Fig. (2).

Uma vez que as inspeções de software são realizadas nas fases iniciais do projeto, tende-se a reduzir o retrabalho nas fases mais avançadas do desenvolvimento, tal redução pode acarretar em aumento da produtividade.



Figura 2 – Processo de inspeção de software. Fonte: (KALLNOWSKI; SPINOLA, 2007).

Inspeções de código-fonte automatizada e verificação formal são técnicas de inspeção de software estáticas que se destinam a somente verificar se o software corresponde à sua especificação, o que não demonstra se o software é útil operacionalmente. Já os testes de software possibilitam a simulação da utilização real do software, o que permite que seja verificado se o produto realiza aquilo a que se destina, isso possibilita que defeitos sejam descobertos analisando as saídas do programa.

Uma vez identificados os defeitos do software, é necessário que seja realizada a correção de tais defeitos. Enquanto os processos de V&V se destinam a determinar a existência ou não de defeitos no software, o *debugging* tem como propósito localizar e corrigir esses defeitos. Um exemplo de um processo de *debugging* pode ser observado na Fig. (3).



Figura 3 – Processo de *debugging*. Fonte: (SOMMERVILLE, 2007).

Em relação ao momento de execução, é importante notar que os processos de V&V e de *debugging* acontecem de forma intercalada, conforme a Fig. (4).

Deve ser ressaltada a importância de se realizar não apenas testes ou inspeções no processo de desenvolvimento de software, mas ambos, pois como afirmam Boehm e Basili (2007), eles podem capturar diferentes tipos de defeitos em diferentes pontos do ciclo de vida de desenvolvimento.

2.4 Automação de testes

De acordo com Lages (2010), "[...] automatizar os testes nada mais é do que repassar para o computador as atividades de testes que normalmente são realizadas de forma manual", sendo essa uma atividade que deve ser iniciada a partir de um processo manual de teste já estabelecido e maduro.



Figura 4 – Processos de V&V e *debugging* acontecem de maneira intercalada.

Segundo Kaner, Falk e Nguyen (apud CAETANO, 2008), "[...] o propósito da automação de testes pode ser resumidamente descrito como a aplicação de estratégias e ferramentas tendo em vista a redução do envolvimento humano em atividades manuais repetitivas". A automação de testes surgiu como forma de apoio aos testes manuais, pois segundo Caetano (2008), "[...] uma abordagem de testes baseada puramente em testes manuais, normalmente não consegue acompanhar as demandas e o volume de testes ao longo do ciclo de vida de desenvolvimento de software".

No processo de desenvolvimento de software, quando finalizadas etapas de codificação, seja durante o ciclo de construção ou após completar-se a implementação, chega-se o momento de testar o produto de software. Muitas organizações trabalham apenas com testes manuais, o que em muitos casos pode ser custoso, em termos de esforço despendido para realizar-se os testes, e não agregar valor ao desenvolvimento. Para lidar com isso, muitas outras organizações investem em ferramentas de automação de testes. Segundo Sommerville (2007), "[...] as ferramentas de teste estão entre as primeiras ferramentas de software a serem desenvolvidas. Atualmente, essas ferramentas oferecem uma grande variedade de recursos e seu uso pode reduzir significativamente os custos de teste."

Dentre as ferramentas de testes automatizados, é possível encontrar os analisadores estáticos e os analisadores dinâmicos. Pfleeger (2004) afirma que a **análise estática** é realizada quando o programa não está sendo executado e que a **análise dinâmica** é feita quando o programa está sendo executado. Cada um dos tipos de ferramentas fornece informações sobre o código em si ou sobre o caso de teste que está sendo executado.

Analisadores estáticos são ferramentas que varrem o código-fonte de um programa e detectam possíveis defeitos e anomalias. A análise estática de código pode ser usada

como parte do processo de inspeção ou como atividade separada do processo de V&V. [Sommerville \(2007\)](#) afirma que a intenção da análise estática é chamar a atenção do inspetor para possíveis anomalias que possam ser encontradas no código-fonte, como variáveis usadas sem serem iniciadas, variáveis não usadas ou dados cujos valores poderiam ficar fora de sua extensão. Tais anomalias podem ser associadas, na maioria dos casos, a erros de programação ou omissões, e, devem enfatizar coisas que poderiam sair erradas quando o programa fosse executado. Ainda segundo [Sommerville \(2007\)](#), é importante enfatizar que essas anomalias, não necessariamente resultam em defeitos de programas.

[Pfleeger \(2004\)](#) agrupa as ferramentas de análise estática em quatro tipos:

- **Analizador de código:** no qual a sintaxe dos componentes é avaliada automaticamente. Os comandos podem ser ressaltados, se a sintaxe estiver errada, se uma construção tende a apresentar defeitos, ou se um item não foi definido;
- **Verificador de estrutura:** essa ferramenta gera um grafo dos componentes submetidos como entrada. O grafo retrata o fluxo lógico, e a ferramenta verifica problemas estruturais;
- **Analizador de dados:** a ferramenta revê as estruturas de dados, as declarações de dados e as interfaces entre os componentes, e registra os links impróprios entre os componentes, definições de dados em conflito e uso ilegal de dados;
- **Verificador de sequência:** a ferramenta verifica as sequências de eventos. Se a codificação estiver na sequência errada, os eventos são ressaltados.

[Sommerville \(2007\)](#) afirma que em linguagens que não possuem regras de tipagem estritas em que os compiladores executam verificações limitadas, como a linguagem C, é fácil para programadores cometerem erros e as ferramentas de análise estática podem descobrir automaticamente um grande número de erros potenciais e reduzir significativamente os custos de teste.

No que diz respeito à análise dinâmica, [Pfleeger \(2004\)](#) afirma que há dificuldade para se testar muitos tipos de sistema pelo fato de diversas operações paralelas serem realizadas simultaneamente, isso impede que todas as condições sejam previstas e sejam gerados casos de teste representativos, e, as ferramentas automatizadas possibilitam que a equipe de testes obtenha os estados dos eventos durante a execução do programa, preservando a visualização das condições.

Algumas ferramentas param quando os pontos de interrupção são atingidos, permitindo que o responsável pelo teste examine o conteúdo da memória ou os valores de itens de dados específicos.

Além das ferramentas para análise do código, ferramentas de análise estática e dinâmica, existem ainda as ferramentas que podem ser utilizadas para automatizar o planejamento e a execução dos testes. Para essas etapas, [Pfleeger \(2004\)](#) define duas categorias de ferramentas:

- **Captura e repetição:** Tais ferramentas capturam as teclas digitadas, as entradas e respostas, à medida que o teste está sendo executado, e compara a saída real com a que é esperada. As discrepâncias são relatadas à equipe de testes, e os dados capturados ajudam a equipe a rastreá-las até chegar às suas raízes;
- **Stubs e drivers:** *Stubs* são programas de propósito especial que simulam a atividade do componente que está faltando. Um *driver* é uma rotina que chama um componente específico e passa um caso de teste para ele.

As ferramentas de automação de testes podem ainda ser integradas com outras, a fim de formar um ambiente de testes. De acordo com [Pfleeger \(2004\)](#), "[...] o teste sempre envolverá o esforço manual requerido para rastrear um problema até sua causa original. A automação auxilia, mas não necessariamente substitui a função humana.

2.5 Metodologias para análise de ROI em Qualidade de Software

Em tecnologia da informação, assim como todas as áreas nas quais decide-se realizar um investimento, é necessário considerar o retorno financeiro proveniente daquele investimento. Caso contrário, o dinheiro poderá ser desperdiçado, colocando as organizações em risco de falência a longo prazo.

Atualmente, muitas empresas têm investido grandes somas em dinheiro na melhoria dos seus processos de software, pois a melhoria de processo deve implicar em melhoria da qualidade. Segundo [Solingen \(2004\)](#),

A melhoria de processos de software visa criar um desenvolvimento de software mais eficaz e eficiente e manter a estruturação e otimização de processos, assumindo que uma organização bem gerida, com um processo de engenharia definido é mais suscetível a produzir produtos que atendam aos requisitos do cliente de forma consistente dentro do cronograma e do orçamento do que uma organização mal gerida sem processo de engenharia definido.

Como foi exposto anteriormente, qualidade de software visa garantir o atendimento às necessidades explícitas e implícitas, visa garantir que o produto esteja aderente aos requisitos do cliente. Dessa forma, pode-se afirmar que melhoria de processo implica em qualidade de software.

Com base em argumentos desse tipo, a questão levantada é: O investimento em melhoria de processo de software vale o custo? O ROI fornece a taxa de retorno de investimento e visa auxiliar as empresas desenvolvedoras de Tecnologia da Informação (TI) no processo de tomada de decisão de investimento.

Segundo [Emam \(2005\)](#),

A quantidade de dinheiro investido na melhoria da qualidade de software é uma decisão de negócios. Uma maneira de informar esta decisão é considerar como qualidade e lucro estão relacionados. Um argumento que pode ser feito é que o investimento adequado em termos de qualidade é o montante que irá maximizar o lucro.

[Emam \(2005\)](#) afirma que analisar o tipo de cliente que adquire o produto pode ser determinante na hora de decidir qual o investimento adequado em qualidade de software. Deve-se olhar para a qualidade que os clientes adquirentes do software esperam e qual o preço que se pode cobrar pelo software. O nível de qualidade de software vai influenciar o preço que os clientes estão dispostos a pagar pelo produto.

Ainda de acordo com [Emam \(2005\)](#), outro conceito importante quando se fala em ROI em qualidade de software é o conceito de receita, que é dado como a multiplicação do preço e da quantidade vendida, conceito este que se relaciona com qualidade. Um produto com baixa qualidade, que possui muitos defeitos, resultará em menos uso e baixo uso significa baixa receita, mas, em contrapartida significa menores custos de pós-lançamento, menores números de relatórios de problemas registrados. Isso gera uma situação de equilíbrio para os produtos de baixa qualidade. Já produtos de alta qualidade resultam em uma maior utilização, que significa maiores receitas, e, em contrapartida, mais defeitos a serem descobertos, aumento nos custos de pós-lançamento. Essa é a situação de equilíbrio para os produtos de alta qualidade, onde a receita irá compensar os custos.

De acordo com a metodologia proposta por [Emam \(2005\)](#), tem-se que lucro é diferente de receita, sendo definido de acordo com a Eq. (2.1).

$$Lucro = [Receita] - [Custos_pré_lançamento] - [Custos_pós_lançamento] \quad (2.1)$$

Onde, os custos de pré-lançamento são definidos como os custos incorridos antes do produto ser repassado ao usuário final. Tais custos são compostos por custos de construção – que estão associados reais de desenvolvimento de software, tais como análise de requisitos, projeto e codificação –, custos de detecção de defeitos – associados ao esforço para procurar defeitos introduzidos durante a construção – e custos de retrabalho – que tratam-se dos custos incorridos para corrigir os defeitos.

Já os custos de pós-lançamento são aqueles contabilizados após o lançamento do produto ao mercado. Os custos de pós-lançamento são compostos pelos custos de retraba-

lho, que nesse caso se deve a correção de defeitos que foram detectados em grande parte por clientes.

As organizações devem se atentar para o fato de que chega um ponto em que os custos de melhoria da qualidade tornam-se muito altos e os benefícios potenciais não vão fazer o custo valer a pena, o que geraria o chamado prejuízo.

Sikka (2004) define ROI como sendo “[...] um conceito geral que se refere a ganhos de investimento de capital, onde os resultados são expressos em proporção de gasto”, e define ainda, que em proporções matemáticas, ROI seria descrito conforme a Eq. (2.2).

$$ROI = \left(\frac{\text{Benefício líquido}}{\text{Custo líquido}} \right) \cdot 100 \quad (2.2)$$

De acordo com Solingen (2004), “[...] calcular benefícios e custos é um pré-requisito para a tomada de decisão de investimento”. Sikka (2004), utiliza uma abordagem baseada em benefícios e custos, e por mais que nem todos sejam facilmente quantificáveis, a metodologia por ele utilizada procura identificar e quantificar todos os benefícios e custos realmente significativos. De acordo com Sikka (2004):

Os benefícios líquidos podem ser diretos, em termos de receita incremental gerada, produtividade adquirida ou despesa economizada, ou indiretos, redistribuição de recursos ou tarefas. Custos líquidos incluem contratações, salários, benefícios, licenciamento de software e despesas gerais e administrativas.

Já Solingen (2004) ressalta que as organizações baseiam o cálculo do custo em estimativa, pois além de considerar os valores citados por Sikka (2004), geralmente utilizam também um valor fixo por hora, o que faz com que o custo se torne um valor com um nível de precisão aceitável muito próximo do real. Solingen (2004) ressalta também que os benefícios podem ser calculados utilizando-se a mesma metodologia que para o cálculo dos custos.

Além disso, é importante ressaltar que os cálculos do ROI devem ser explícitos e não baseados em intuição, de modo a evitar avaliações intuitivas incorretas. Sem os valores de benefício e de custo, é impossível decidir se o investimento vale o custo.

Ainda segundo Solingen (2004), é necessário se concentrar nos impactos na produtividade e no tempo de colocação no mercado. Segundo Diaz e Sligo (1997), “[...] O verdadeiro custo benefício ocorre quando os projetos terminam mais cedo, o que permite aplicar mais recursos de engenharia para a aquisição e desenvolvimento de novos negócios”.

A metodologia de ROI sugerida por Solingen (2004) calcula o ROI dividindo o lucro do investimento pelo investimento, conforme a Eq. (2.3).

$$ROI = \frac{Benefício - Custo}{Custo} \quad (2.3)$$

Em seu trabalho, Rico (2004) apresenta várias metodologias para a análise de ROI, dentre as quais uma metodologia voltada para o processo de inspeção de software, o que define como “[...] uma forma de medir, quantificar e analisar o seu valor econômico”. Tal metodologia envolve um processo de seis partes, que consiste em: estimar benefícios, custos, relação custo/benefício, ROI, Valor Presente Líquido (VPL) e ponto de equilíbrio.

Benefícios podem ser definidos como o montante resultante de método de melhoria da qualidade. É o valor econômico resultante de um processo com melhorias implementadas. A fórmula para cálculo dos benefícios é dada pela Eq. (2.4).

$$Benefício = \sum_{t=1}^n Benefício_i \quad (2.4)$$

Custo é definido como o valor gasto com a implementação de uma melhoria, é considerado uma consequência econômica da decisão de se investir em melhoria. E sua fórmula é dada pela Eq. (2.5).

$$Custo = \sum_{t=1}^n Custo_i \quad (2.5)$$

Já a relação custo/benefício é definida simplesmente como a razão entre os custos e os benefícios, conforme a Eq.(2.6).

$$R_{C,B} = \frac{Benefícios}{Custo} \quad (2.6)$$

ROI é dado como a quantidade de dinheiro que retorna a partir de um investimento. Na metodologia abordada por Rico (2004), o ROI é calculado conforme a Eq.(2.7).

$$ROI = \left(\frac{Benefício - Custo}{Custo} \right) \cdot 100\% \quad (2.7)$$

Nesse caso, os custos são primeiro subtraídos dos benefícios antes de serem divididos pelos custos. Dessa forma, quantifica-se os benefícios verdadeiros.

O VPL é definido como o valor econômico do dinheiro atual no futuro, é uma forma de quantificar o valor do dinheiro de modo a permitir determinar corretamente o quanto o dinheiro valerá no futuro. É definido de acordo com a Eq. (2.8).

$$VPL = \frac{Benefício}{(1 + taxa\ de\ inflação)^{número\ de\ anos}} \quad (2.8)$$

O ponto de equilíbrio é um valor numérico que diz que os benefícios ultrapassam ou excedem os custos, é quando começa-se a fazer um lucro acima do nível de despesas. Mesmo que não se relacione com os outros conceitos apresentados (custo, benefício, relação custo/benefício, ROI e VPL), é indispensável para definir quando os benefícios serão alcançados. É dado pela Equação (2.9), que a título de exemplo leva em consideração a produtividade.

$$PE = \frac{Custo}{1 - \left(\frac{ProdutividadeAntiga}{NovaProdutividade} \right)} \quad (2.9)$$

De acordo com [Emam \(2005\)](#),

Fornecedores de software recebem pagamentos independentemente da confiabilidade do software entregue e são frequentemente dados recursos adicionais para corrigir problemas de sua própria criação: muitas vezes é lucrativo entregar produtos de má qualidade, mais cedo, em vez de produtos de alta qualidade mais tarde. Correções de bugs são geralmente embalados como novos lançamentos e vendidos para gerar mais receita.

Porém, existe um ônus quando decide-se entregar produtos ao cliente que possuam *bugs* que necessitam ser corrigidos. As equipes que acabam por apagar muitos incêndios e diminuem a capacidade produtiva de sua empresa. Entregando baixa qualidade, pode-se acabar com a reputação de uma empresa. No entanto, vale ressaltar que entregar um software de qualidade, não significa entregar um software com “zero defeito”.

Segundo [Emam \(2005\)](#),

Em vez de apontar para “zero defeito”, pode-se avaliar os custos e benefícios de investimentos específicos em qualidade de software. Então, com base em argumentos econômicos, uma decisão é tomada sobre onde e como melhorar a qualidade.

De acordo com a metodologia proposta por [Emam \(2005\)](#), os defeitos devem ser classificados em maiores e menores. Defeitos maiores influenciam significativamente o comportamento e as características do software que são visíveis para o cliente. Já os defeitos menores, são defeitos que afetariam as características internas do produto como complexidade, facilidade de compreensão e modificação e eficiência.

[Emam \(2005\)](#) utiliza o termo defeitos para se referir a falta, erro, falha, relatório de problema, redescoberta e má correção. Abaixo o significado de cada um dos termos:

- **Falta:** causa de uma falha, se refere ao aspecto físico. Por exemplo: código incorreto ou faltando, defeito de hardware, por exemplo;
- **Erro:** ação humana que causa um defeito ao ser inserido no software;

- **Falha:** resultado incorreto produzido pelo software. Pode ser funcional – o software não implementa a funcionalidade adequada – ou não funcional – falha de desempenho;
- **Relatório de problema:** podem ser abertos em resposta a uma falha tanto por usuários internos quanto por usuários externos à organização, uma vez que o software já tenha sido lançado. Trata-se da descrição de uma falha;
- **Redescoberta:** ocorre quando alguém identifica uma falha ou defeito que já foi descoberto anteriormente;
- **Má correção:** quando um defeito é corrigido, a correção pode acarretar um ou mais novos defeitos.

Porém, na metodologia proposta por [Emam \(2005\)](#), todos os termos acima definidos são tratados como defeitos, para efeito de generalização.

Para se obter o dado referente ao número de defeitos, deve-se considerar que devem ser contados tanto defeitos abertos quanto fechados, pois contar apenas defeitos abertos ou fechados, pode dar uma margem imprecisa sobre a contagem de defeitos. A quantidade total de defeitos em um sistema é a soma dos defeitos contados por módulo.

[Emam \(2005\)](#) sugere utilizar o cálculo de densidade de defeitos, para uniformizar as diferentes interpretações que podem ser encontradas nos diferentes módulos que compõem um sistema, mesmo não sendo a medida ideal de qualidade, porque a relação entre defeitos e tamanho não é linear, mas ainda assim é melhor do que contar cruamente os defeitos. A densidade de defeitos é dada pela Eq. (2.10).

$$\text{Densidade de Defeitos} = \frac{\text{Número de defeitos encontrados}}{\text{Tamanho}} \quad (2.10)$$

Na fórmula, o tamanho pode ser medido por milhares de linhas de código ou por pontos de função, por exemplo.

Com a intenção de demonstrar quão custosa a falta de qualidade pode ser para o cliente final, [Emam \(2005\)](#) apresenta um modelo para calcular a relação entre qualidade de software e o custo de propriedade do software. Durante a aquisição do software, os custos do cliente são compostos por três elementos: Custos de pré-compra, custos de instalação e custos de pós-compra.

Além dos custos para os clientes, existem, segundo [Emam \(2005\)](#), os custos para os desenvolvedores, que podem ser distribuídos em categorias de custos:

- **Custos de pré-lançamento:** são os custos incorridos antes do produto ser lançado, que envolvem:

Custos de construção: que consistem no esforço associado com as atividade de software reais de desenvolvimento, tais como análise de requisitos, projeto e codificação;

Custos de detecção de defeitos: compreendem o esforço para procurar defeitos introduzidos durante a construção;

Custos de retrabalho: referem-se aos custos de correção de defeitos encontrados durante os testes e inspeções.

- **Custos de pós-lançamento:** são custos associados à retrabalho, que no pós-lançamento são relativos a correção de defeitos que são detectados em grande parte por clientes.

Diante do detalhamento dos custos, pode-se perceber que os custos de retrabalho estão associados a ambas categorias de custos de desenvolvimento, o que permite inferir que reduzir os custos de retrabalho é a melhor maneira de reduzir os custos do projeto. É possível alcançar essa redução reduzindo o número de defeitos que realmente são introduzidos no software e melhorando a forma de detecção de defeitos no software.

É importante ressaltar que corrigir defeitos no pós-lançamento sai muito mais caro do que encontrá-los e corrigi-los no pós-lançamento, uma vez que, os custos de defeitos encontrados e corrigidos nas fases iniciais do projeto são relativamente mais baixos.

A metodologia de cálculo de ROI proposta por [Emam \(2005\)](#) diz que, do ponto de vista do desenvolvedor, dois importantes benefícios podem advir da implementação de práticas de qualidade: dinheiro e tempo. Dessa forma, sugere que dois tipos de ROI sejam calculados, redução de custos (*cost saving*), que foca na economia de custos; redução de prazo (*schedule reduction*), que foca na economia do tempo gasto no ciclo de vida do software.

[Emam \(2005\)](#) apresenta dois modelos de cálculo de ROI considerando a redução de custos, porém um deles não se aplica ao contexto em questão porque não considera os benefícios da qualidade de software, trata-se de uma medida tradicional de ROI. Dessa forma, o modelo baseado no trabalho de [Kusumoto \(1993\)](#), que é interpretado como as economias globais do investimento do projeto, ou seja, trata-se de uma medida de poupança, torna-se mais adequado e é dado pela Eq. (2.11):

$$ROI = \frac{\text{Custo Economizado} - \text{Investimento}}{\text{Custo Original}} \quad (2.11)$$

A redução de prazo (SCEDRED do inglês *schedule reduction*) é formulada como uma fração do cronograma original, conforme a Eq. (2.14):

$$SCEDRED = \frac{\text{Cronograma Original} - \text{Novo Cronograma}}{\text{Cronograma Original}} \quad (2.12)$$

Substituindo a equação do COCOMO II para a programação, tem-se a Equação (2.15), que expressa a redução de cronograma diretamente em termos de redução de custo.

$$SCEDRED = 1 - (1 - ROI)^{0.28 + \left(0.002 \times \sum_{j=1}^5 SF_j\right)} \quad (2.13)$$

Onde:

SF_j se refere a uma série de 5 fatores de uma escala usada para ajustar a programação. Fatores que podem ser observados na Tab. (3).

Tabela 3 – Definição da escala de 5 fatores requerida para redução de prazo

Fator da escala	Definição
Precedência (PREC)	Trata-se de uma medida da experiência da organização com sistemas similares. Se um produto é similar a diversos produtos desenvolvidos previamente, o PREC é alto.
Flexibilidade de desenvolvimento (FLEX)	Mede a habilidade das organizações para acomodar requisitos e interfaces preestabelecidos com o intuito de facilitar o esforço de desenvolvimento de software.
Arquitetura/Resolução de riscos (RESL)	Mede a habilidade das organizações de realizar revisões de projeto bem como outros fatores mitigadores de risco.
Coesão do time (TEAM)	Diz respeito às fontes de turbulência e entropia no projeto devido às dificuldades de sincronização dos <i>stakeholders</i> do projeto.
Maturidade do processo (PMAT)	Maturidade medida pelo SW-CMM ou equivalente.

O primeiro passo para se realizar a avaliação de ROI é estabelecer a *baseline* para o projeto e essa *baseline* consiste em dois passos:

Passo 1. Um modelo de processo do ciclo de vida de detecção de defeitos do projeto;

Passo 2. Dados que caracterizem as atividades do ciclo de vida de detecção de defeitos.

O Passo 1 consiste em desenvolver um modelo de processo que documenta uma série de atividade por todo o ciclo de vida que elimina os defeitos.

O Passo 2 consiste em caracterizar as atividades no modelo, determinando os seguintes valores para cada uma das atividades:

- A eficácia da atividade, ou seja, a proporção de defeitos que essa atividade pode detectar.

- O custo para encontrar e corrigir um único defeito usando a atividade.

Para simplificar, considera-se que não é necessário caracterizar qualquer atividade pós-lançamento, e, não é necessário caracterizar qualquer atividade antes da atividade para a qual você deseja calcular o ROI.

Existem três formas de se coletar os dados para caracterizar as atividades do modelo, sendo elas:

1. Coletar os dados por mineração em bancos de dados de métricas existentes dentro da organização, caso a organização possua um programa de medição.
2. Utilizar dados baseados em referências internacionais, que consistem em dados de muitos projetos do mundo todo, caso a organização não possua um programa de medição.
3. Entrar em contato com o fornecedor e pedir a eficácia e os dados de custo. Ou, caso seja necessário, em seguida entrevistar os clientes existentes ou os membros de um grupo de usuários para se ter uma ideia da eficácia e do custo por defeito.

Depois de estabelecida a *baseline* já é possível realizar a avaliação de ROI, utilizando as equações e modelos apresentadas por [Emam \(2005\)](#).

Caso haja alguma incerteza associada ao resultado, será necessário realizar uma análise de sensibilidade, que consiste em analisar a sensibilidade a mudanças para analisar o efeito sobre o ROI, atribuindo a uma variável incerta uma gama de valores possíveis ao invés de um único valor. Há que se lembrar que cálculo de ROI, redução de custo e redução de cronograma são estimativas e, portanto, há incertezas associadas a essas estimativas, decorrentes de variáveis que não são contabilizadas.

2.6 Considerações Finais

O referencial teórico apresentado consolida conceitos variados que são importantes para a compreensão do que tratará o trabalho proposto. Quanto à qualidade de software, tem-se que se destina a garantir conformidade ao produto de software, porém sua definição depende do contexto a que se aplica.

Já no que se refere à medição de software, pode-se afirmar que a função da medição é permitir mensurar os atributos dos processos e dos produtos de software. A medição de software é importante para apoiar a gestão dos projetos, pois norteia o processo de tomada de decisão, permitindo mapear pontos de falha ou sucesso.

Quanto à V&V, é importante fazer a distinção dos termos. De forma mais simples, pode-se afirmar que a verificação está relacionada com o processo, pois visa garantir a qualidade do processo de engenharia de software, enquanto que a validação está relacionada com o produto, está focada na garantia da qualidade do produto de software. Como processos de V&V podem ser destacados os testes – que verificam o comportamento operacional do software – e as inspeções – que verificam os produtos de projeto do software.

No que diz respeito a automação de testes, tem-se que pode-se diminuir os custos de realização de testes automatizando-os e que essa diminuição de custos pode ser facilitada pelos muitos tipos de ferramentas de automação existentes.

Um processo de garantia da qualidade não garante que serão produzidos softwares livres de defeitos, mas cria na equipe de desenvolvimento a consciência de produzir softwares com a mínima quantidade possível de defeitos.

Um passo importante para o processo de tomada de decisão em qualquer organização desenvolvedora de software é considerar o retorno financeiro advindo dos investimentos realizados. De forma geral, para que um ROI seja satisfatório para uma organização deve haver equilíbrio entre receita - que é o produto das vendas - e custos. A receita deve compensar os custos. Essa avaliação vai permitir avaliar se os custos compensam o investimento.

O trabalho proposto no capítulo a seguir permitirá que os conceitos definidos sejam observados. Será possível observar a relação entre eles e os resultados serão obtidos através da correta utilização dos mesmos.

3 Aplicação da Metodologia

O presente capítulo, denominado "Aplicação da Metodologia" está se encontra dividido em seções, conforme descrito a seguir:

A seção 3.1 apresenta o que é a metodologia de trabalho adotada na execução deste Trabalho de Conclusão de Curso

A seção 3.2 apresenta o que é o Centro de Qualidade e Testes de Software (CQTS), seus objetivos e sua composição, permitindo que se conheça o contexto ao qual se aplicará a metodologia.

Na seção 3.3, é possível visualizar, observar e entender quais os processos que compõem o CQTS, como funcionam e se relacionam entre si. A descrição de tais processos permitirá entender o fluxo de trabalho e o "caminho" que os cadernos de teste percorrem dentro de laboratório.

A seção 3.3 traz a justificativa para a escolha da metodologia, permitindo saber as razões pelas quais a metodologia em questão é a mais adequada para o contexto do CQTS.

Na seção 3.4, é possível conhecer um pouco de como se dá o processo de automação de testes no CQTS e como é realizado o registro dos resultados dos testes.

Na seção 3.5, é realizada uma adaptação da metodologia ao contexto do CQTS, uma vez que o contexto proposto pelo autor é um pouco diferente do contexto do CQTS.

A seção 3.6 apresenta os resultados de execução dos testes manuais e automatizados, apresentando-os em tabelas. A seção apresenta também o registro de esforço da execução dos dois tipos de teste e da construção de *scripts*.

Na seção 3.7, é possível acompanhar o passo-a-passo da aplicação da metodologia e dos cálculos realizados para alcançar os resultados esperados do trabalho.

E, finalmente, a seção 3.8, apresenta a interpretação dos resultados encontrados.

3.1 Metodologia de Trabalho

A proposta inicial do presente Trabalho de Conclusão de Curso consistia em aplicar a metodologia proposta por [Emam \(2005\)](#) no contexto do CQTS, por ser baseada em contagem de defeitos. Tal característica da metodologia, permitiria que houvessem insumos suficientes para a aplicação da metodologia.

A intenção inicial era realizar a avaliação de ROI obedecendo aos seguintes passos:

1. Identificação das atividades que serão observadas do ciclo de vida de detecção de defeitos do CQTS.
2. Definição das medidas que serão utilizadas para viabilizar a aplicação da metodologia no contexto proposto.

O primeiro passo consistia basicamente em mapear as atividades de detecção de defeitos do CQTS para conhecer o funcionamento de tal processo. E o segundo passo consistia em identificar as medidas que seriam utilizadas e, no caso em questão, adaptadas, de modo a viabilizar a aplicação da metodologia no contexto proposto.

A aplicação da metodologia obedeceu o fluxo apresentado na Fig. (5).

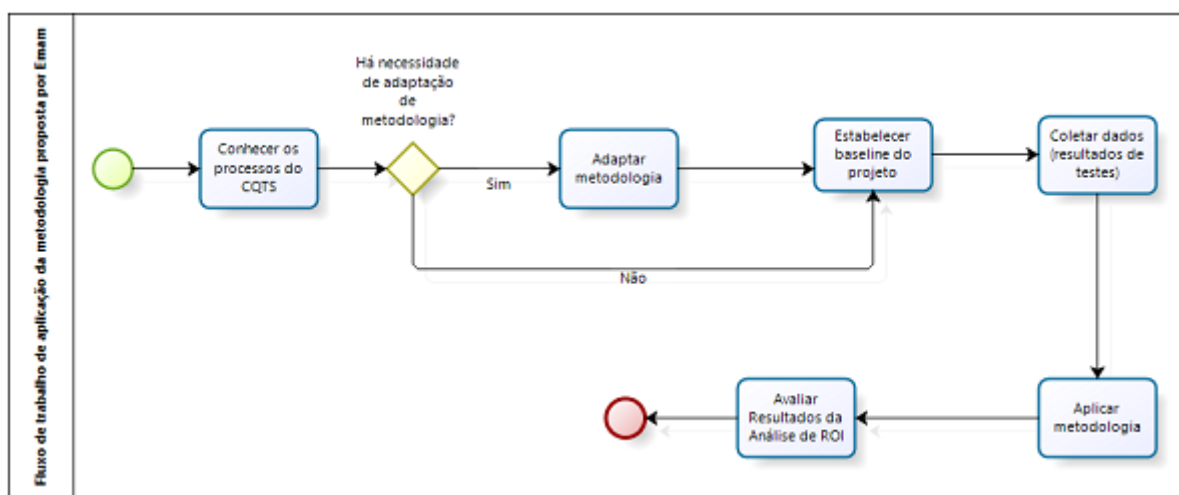


Figura 5 – Fluxo utilizado para aplicação da metodologia

Na atividade denominada **Conhecer os processos do CQTS**, foi realizado um mapeamento dos processos existentes no laboratório a fim de entender o fluxo de processos no laboratório.

Após a realização da etapa supracitada, foi verificada a necessidade se realizar uma adaptação de metodologia, o que gerou a atividade denominada **Adaptar metodologia**.

Na atividade denominada **Estabelecer *baseline* do projeto**, definiu-se o caderno de testes que seria trabalhado.

Logo em seguida foram realizadas as atividades **Coletar dados (resultados dos testes)**, que consistiu em executar os testes manuais e automatizados e coletar seus resultados, e **Aplicar metodologia**, em que se deu a aplicação das fórmulas sugeridas pelo autor e adaptadas.

Por fim, foi realizada a atividade **Avaliar os Resultados da Análise de ROI**, na qual foram interpretados os resultados encontrados após a aplicação da metodologia.

3.2 O que é o Centro de Qualidade e Testes de Software?

O Centro de Qualidade e Testes de Software (CQTS) possui ênfase em pesquisa e desenvolvimento em qualidade de software e, atualmente, tem atuado com testes de aplicativos para dispositivos móveis de um parceiro do CQTS. Seus objetivos consistem em:

- Realizar pesquisa e desenvolvimento na área de Qualidade de Software;
- Realizar experimentos em ambientes reais e controlados de desenvolvimento para validar os produtos resultantes da pesquisa;
- Criar uma infraestrutura de qualidade de software que poderá ser instanciada de acordo com o tipo, tamanho e outras características do software a ser desenvolvido; e,
- Colaborar para a formação teórico-prática do estudante de graduação em Engenharia de Software da FGA.

A maior vantagem do CQTS é possibilitar a ambientação do aluno da Faculdade do Gama (FGA) ao contexto de empresa, uma vez que se envolvendo nos projetos do CQTS o aluno terá contato com projetos reais de seus parceiros, e dessa forma, irá adquirir experiência, o que poderá torná-lo mais preparado para os desafios que se darão após a graduação.

As equipes envolvidas nos projetos são sempre compostas por alunos e professores de Engenharia de Software com perfil de atuação nas áreas de qualidade e teste que poderão desempenhar diferentes papéis ao longo da execução dos projetos.

3.3 Processos do CQTS

Para que fosse possível conhecer o funcionamento do CQTS, foi necessário realizar um mapeamento dos processos utilizados no laboratório. Durante essa etapa, foi possível detectar dois processos principais, o “Gerenciamento de Demanda” e o “Processo de Teste”.

O processo de “Gerenciamento de Demanda” é dividido nos fluxos de “Recebimento e alocação de demanda” e “Acompanhamento de demanda”, conforme pode ser visualizado na Fig. (6).

Já o “Processo de Teste” é dividido nos fluxos de “Planejamento da Demanda”, “Execução de Testes”, “Construção de Testes” e “Automação de Testes”, conforme apresentado na Fig. (7).

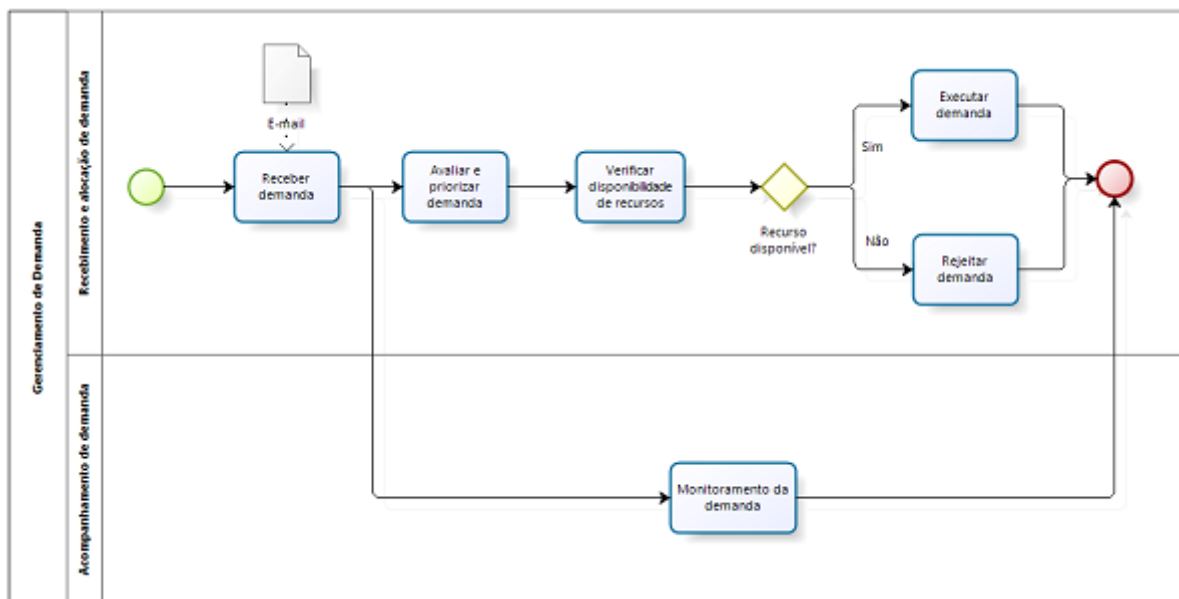


Figura 6 – Processo de Gerenciamento de Demanda do CQTS.

A descrição detalhada dos processos pode ser visualizada no Anexo A.

No caso do presente Trabalho de Conclusão de Curso, optou-se por analisar o impacto financeiro da automação de testes, pelo fato de a automação de testes ser muito importante para o CQTS, pois um de seus negócios é a execução de testes, e pelo resultado da aplicação da metodologia nesse contexto pode fornecer *feedbacks* importantes ao CQTS.

A escolha por trabalhar com a metodologia proposta por Emam (2005) se deu pelo fato de considerar a contagem dos defeitos encontrados durante os testes de software e se encaixar com o propósito do CQTS, que é o de testar software e encontrar defeitos, provendo, com isso, melhoria de qualidade.

Após a análise dos fluxos de processos do CQTS, foi observada a necessidade de que a metodologia fosse adaptada à realidade do laboratório, pois foi desenvolvida considerando projetos de desenvolvimento de software e abrange muitas características de desenvolvimento e correção de defeitos pós-teste. Pelo fato do CQTS ser voltado à execução de testes apenas, e não tratar de correção de defeitos e disponibilização de sistemas ao cliente, além de outras características inerentes aos processos de desenvolvimento de software, propõe-se uma adaptação da metodologia, para que a mesma fique voltada para o contexto de fábrica de teste.

3.4 O CQTS e a automação de testes

No CQTS, as duas modalidades de teste - manual e automatizado - são realizadas e verifica-se a possibilidade de automação para cada um dos casos de teste. Geralmente, os casos de teste considerados automatizáveis são aqueles que são fixos, ou seja, que se

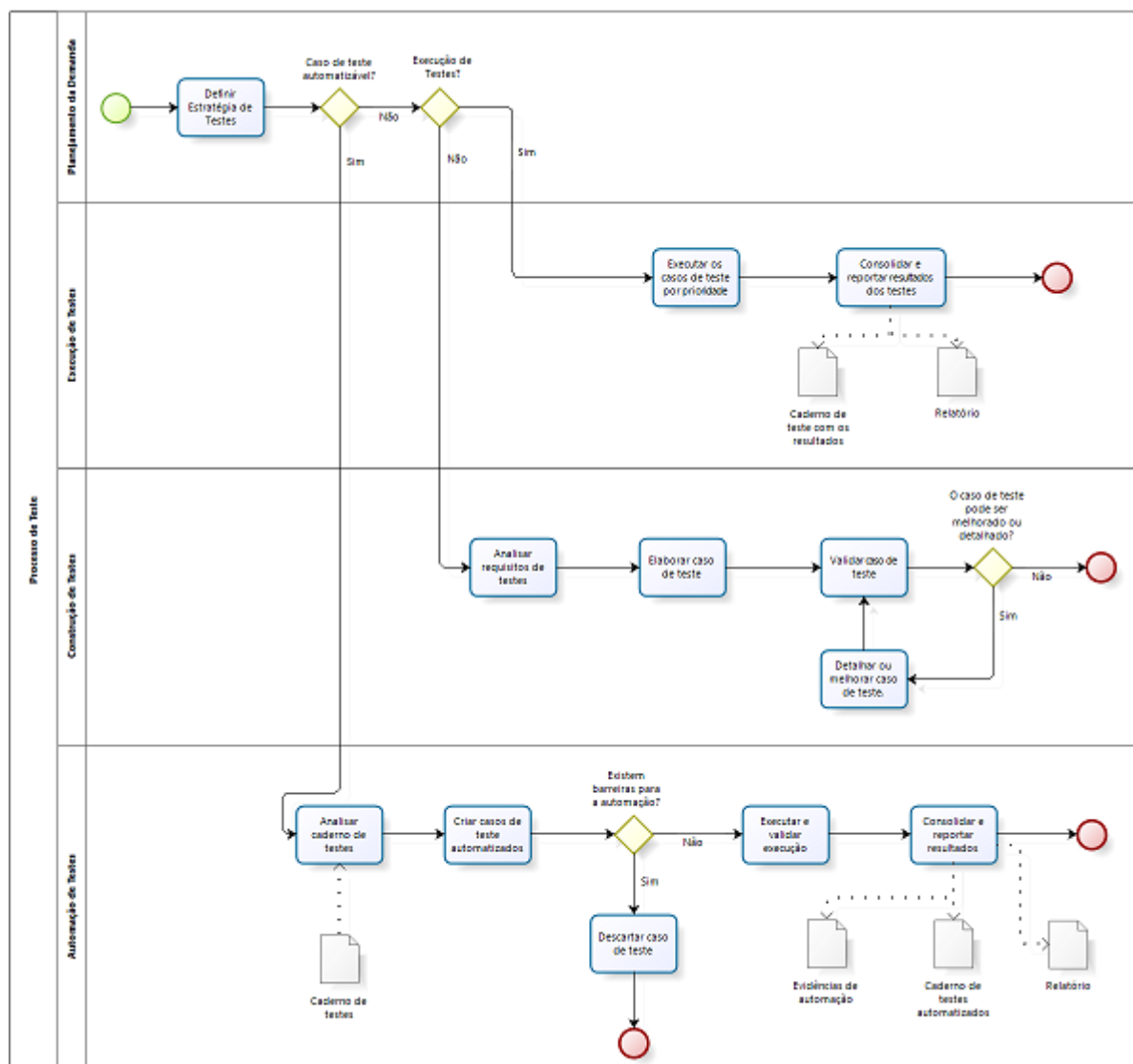


Figura 7 – Processo de Testes do CQTS.

repetem, ou seja, a execução não muda de um caderno de testes para o outro, que é o caso dos testes de *wi-fi* ou testes de câmera, por exemplo. Anteriormente, existia a possibilidade de um caso de teste ser considerado semi-automatizável, que eram os casos em que era necessário finalizar os testes manualmente. Porém, atualmente, já é possível cobrir totalmente um caso de teste considerado automatizável.

As aplicações testadas no laboratório do CQTS são em sistema Android, que é o Sistema Operacional (SO) que roda nos aparelhos do parceiro do CQTS e, para que seja possível realizar os testes, o ambiente de testes deve possuir o SDK do Android, o Java Development Kit (JDK) da versão 6.3 em diante e o Python 2.7.

A ferramenta de testes utilizada para a automação de testes no CQTS é o MonkeyRunner. Trata-se de uma ferramenta que vem no próprio SDK do Android e oferece uma API e um ambiente de execução para testes escritos em Python. A API permite conectar-se a um dispositivo, instalar ou desinstalar aplicações, executá-las, obter cap-

turas de tela, comparar imagens para verificar se a tela contém o que se espera após a execução de certos comandos, e usar um conjunto de testes para testar uma aplicação. Inclusive, a comparação de telas é o parâmetro para obtenção dos resultados dos testes realizados no CQTS. Para que um teste "passe", a tela obtida deve ser exatamente igual ao esperado, caso contrário, o teste falhará.

Após a execução dos testes, o programa gera algumas pastas com resultados dos testes, dentre as quais uma pasta chamada "*snapshots*" que traz as imagens de tela capturadas após os testes que o programa utilizará como parâmetro de comparação para gerar os resultados apontando falha ou sucesso e uma pasta chamada "*reports*" que trará o resumo dos resultados da execução dos testes.

Os resultados provenientes dos testes manuais são registrados em planilhas excel e, geralmente, são compostos pelos seguintes campos:

Case de Teste (*Test Case*): Identificador do caso de teste;

Item de Teste (*Test Item*): Descrição de qual item deve ser testado, distinguindo se trata-se de um item de *hardware* ou *software*, qual parte daquele item deve ser testada e a definição dos itens de teste;

Ação (*Action*): Define quais ações devem ser realizadas a cada procedimento de teste;

Passos (*Steps*): Apresenta o passo-a-passo do teste a ser realizado;

Resultado Esperado (*Expected Result*): Apresenta o resultado esperado após a execução dos testes, tendo sido respeitados os procedimentos estabelecidos;

Resultado (*Result*): Apresenta brevemente o resultado obtido após a execução dos testes, que podem ser: OK, caso o caso de teste tenha passado; NOK, caso tenha havido falha; NOT TESTED, se por alguma razão o caso de teste não possa ter sido testado; e, NOT APPLY, caso a execução do teste não seja aplicável ao caso de teste;

Observação (*Observation*): Registra o resumo dos resultados finais e traz quaisquer observações associadas ao resultado do teste.

A visualização de como fica disposta a planilha, pode ser observada na Fig. (8).

Já os resultados provenientes dos testes automatizados, conforme dito anteriormente, ficam registrados numa pasta chamada "*reports*", que é gerada automaticamente após a execução dos testes.


Os itens definidos na planilha de testes manuais, tais como, Caso de Teste, Item de Teste, Ação, Passos e Resultado Esperado são definidos dentro do próprio *script* de teste, e o relatório gerado após a execução dos testes trará o item Resultado e as observações são relatadas pelo próprio testador, após realizar as análises das imagens na pasta *snapshots*.

SOFTWARE SMOKE TEST CHECKLIST


Date:	15/09/2015
Product:	Tablet
Revision:	POSITIVE - SYSTEM
Author:	
HW Model:	
Android Version:	
SW Version:	
Samples:	Amostra 1

HW ITEM


SW ITEM




☐



☒



☐



☐

*C = Customization

SCORE
92%

NOT TESTED
 48

NOT APPLY
 10

TEST CASE	TEST ITEM	ACTION	STEPS	EXPECTED RESULT	RESULT	OBSERVATION
1	QUADRO GEOMOS Benchmark	Executar teste de velocidade e opção Full Benchmark e verificar a pontuação média	1. Abra Quadrant Advanced benchmark. 2. Execute teste de velocidade e opção Full Benchmark. 3. Calcule a pontuação média	Total: 1823 CPU: 4300 MEM: 1000 IO: 500 2D: 100 3D: 280	NGK	Total: 3454 ; CPU: 3561 ; Mem: 4323 ; IO: 1079 ; 2D: 547 ; 3D: 2511
2	CPU Benchmark	Avaliar desempenho de processamento do dispositivo - CPU	1. Abra Quadrant Advanced benchmark e verifique CPU	>= 1GHz	NOT TESTED	Não possui o APK

Figura 8 – Composição da planilha para registro dos resultados dos testes no CQTS.

Exemplos de como é o relatório gerado após a execução dos testes podem ser visualizados nas figuras (9) e (10).

```

ct125
TEST ITEM: [REDACTED] APKS - Sincronize
ACTION: Abra a APK e verificar se ele tem problemas críticos
STEP:    1. Abra o APK
         2. Utilize o apk por 02 minutos
EXPECTED: 1. O APK é aberto com sucesso e nenhum problema é apresentado
Resultado Obtido: Nao Passou -- O apk nao abriu corretamente.
               Verifique a pasta snapshots para consultar o erro.
Tempo de Execucao: 00:01:19
  
```

Figura 9 – Relatório de resultados dos testes automatizados por caso de teste.

```

Tempo Total de Execucao: 00:36:08
Casos de Teste:
Quantidade Total: 25
Sucessos: 4
Semi-Automatizado: 1
Falhas: 18
Erros: 2
  
```

Figura 10 – Relatório de resultados dos testes automatizados por suíte de teste.

3.5 Adaptação da metodologia

Observando o modelo completo de cálculo de ROI proposto por Emam (2005), foi possível perceber que foi realizada a separação dos custos em duas categorias: “custo total de projeto” e “custos de detecção de defeito”. Nos custos categorizados como “custo total de projeto”, os custos são definidos de acordo com a Tab. (4).

Tabela 4 – Definição dos elementos de custo usados em um projeto de software. Fonte: (EMAM, 2005)

Elemento de custo	Notação
Custos gerais e de correção	C_1
Custos de construção	C_2
Custos de detecção de defeitos (pré-lançamento)	C_3
Custos de retrabalho (pré-lançamento)	C_4
Custos de retrabalho (pós-lançamento)	C_5
Custos de implementação de novas <i>features</i> (pós-lançamento)	C_6
Custos totais de pré-lançamento	$C_A = C_1 + C_2 + C_3 + C_4$
Custos totais de pós-lançamento	$C_B = C_5 + C_6$

Por se tratar de uma metodologia voltada para desenvolvimento de software, e o contexto do CQTS ser de fábrica de software, foi necessária uma adaptação dos custos envolvidos no cálculo do ROI e uma adaptação da metodologia em si.

Na análise do ROI em automação de testes, foi necessária uma adaptação do que se chama de “custo total do projeto” para “custo total de testes automatizados” e “custo total de testes manuais”. Assim, foi preciso definir tais custos e adequá-los à metodologia proposta por Emam (2005).

De acordo com Jayachandran (2005), os seguintes custos fixos são aplicáveis a automação de testes:

- Custo de licenciamento;
- Custo de treinamento;
- Custo de infraestrutura.

O custo de licenciamento está associado aos valores destinados à aquisição de licenças de ferramentas para automação de testes. Cabe ressaltar que as licenças podem ser compradas, locadas ou *free*.

O custo de treinamento está associado ao treinamento para utilização do ambiente de testes, inclui o tempo gasto para treinar cada membro da equipe de testes responsável por trabalhar com automação de testes.

O custo de infraestrutura diz respeito aos equipamentos necessários para a execução dos testes automatizados. Existem ainda, custos variáveis associados, que incluem:

- Custo de projeto;
- Custo de manutenção.

O custo de projeto é o custo advindo dos vários ciclos de teste do projeto e geralmente depende do cronograma do projeto e é dado pela Eq. (3.1).

$$\text{Custo de projeto} = \text{número de horas por pessoa} \times \text{custo de testador por hora} \quad (3.1)$$

O custo de manutenção está associado à contratação de serviços como consultoria para inspeção de ferramenta, manutenção de infraestrutura, atualização de ferramenta, suporte técnico, etc. Porém, como a ferramenta utilizada no CQTS é uma ferramenta *free*, tais custos de manutenção não são incluídos. Dessa forma, o único custo variável a ser considerado é o custo de projeto.

Considerando as definições supracitadas dos custos relativos ao “custo de testes automatizados”, tem-se os resultados os elementos de custo exibidos na Tab. (5).

Tabela 5 – Definição dos elementos de custo associados a automação de testes

Elemento de custo	Notação
Custo de licenciamento	CA_1
Custo de treinamento	CA_2
Custo de infraestrutura	CA_3
Custo de projeto	CA_4
Custo total de testes automatizados	$CA_A = CA_1 + CA_2 + CA_3 + CA_4$

Ainda de acordo com Jayachandran (2005), os custos de licenciamento e treinamento não se associam ao “custo total dos testes manuais”, uma vez que não é necessária uma ferramenta para realização de tais testes, porém tal como no “custo total dos testes automatizados” são considerados os custos de infraestrutura e de projeto, ficando definidos os custos de testes manuais conforme a Tab. (6).

Tabela 6 – Definição dos elementos de custo associados a execução de testes manuais

Elemento de custo	Notação
Custo de infraestrutura	CM_1
Custo de projeto	CM_2
Custo total de testes manuais	$CM_M = CM_1 + CM_2$

O que na metodologia de Emam (2005) era chamado de “custo total de projeto”, será denominado de “custo total de teste” e será definido conforme a Eq. (3.2).

$$\text{Custo total de teste} = CA_A + CM_M \quad (3.2)$$

Na metodologia proposta por Emam (2005), há ainda a notação usada para o esforço de encontrar e corrigir defeitos, conforme a Tab. (7).

Tabela 7 – Definição dos elementos para determinar o esforço de encontrar e corrigir defeitos. Fonte: (EMAM, 2005)

Elemento de esforço	Notação
Esforço para criar ou recriar uma falha ou detectar um defeito em uma atividade do ciclo de detecção de defeitos f	$E_{1,f,i}$
Esforço para isolar um defeito em uma atividade de detecção de defeitos f	$E_{2,f,i}$
Esforço para corrigir um defeito em uma atividade de detecção de defeitos f	$E_{3,f,i}$
Esforço para finalizar um defeito (reteste, documentação e empacotamento) em uma atividade de detecção de defeitos f	$E_{4,f,i}$
Esforço total para encontrar e corrigir um defeito em uma atividade de detecção de defeitos f	$E_{f,i} = E_{1,f,i} + E_{2,f,i} + E_{3,f,i} + E_{4,f,i}$

Onde f é usado para denotar o tipo de atividade de detecção de defeitos e i a quantidade de vezes que essa atividade foi executada.

Uma vez que os defeitos encontrados não serão corrigidos pela equipe do CQTS, a Tab. (7) deve ser reescrita, conforme a Tab. (8).

Tabela 8 – Adaptação dos elementos para determinar o esforço de encontrar e corrigir defeitos.

Elemento de esforço	Notação
Esforço para construção de <i>script</i>	$E_{1,f}$
Esforço para detectar um defeito em uma atividade do ciclo de detecção de defeitos f	$E_{2,f}$
Esforço total para encontrar defeito em uma atividade de detecção de defeitos f	$E_{f,i} = E_{1,f} + E_{2,f}$

Não será utilizado o cálculo de densidade de defeitos apresentado na equação (2.10), pois é uma equação que leva em consideração do tamanho do software construído. Seria possível considerar o tamanho dos caderno de testes em casos que fossem considerados muitos cadernos de testes, porém como trata-se de apenas um caderno, a utilização desse dado poderia impactar diretamente o resultado final da aplicação da metodologia. Dessa forma, será considerada a contagem crua de defeitos.

Originalmente, o modelo financeiro de ROI consiste de três elementos, conforme a Eq. (3.3).

$$ROI\ Financeiro = [ROI\ de\ pré_lançamento] + [ROI\ de\ pós_lançamento] + [ROI\ do\ reuso] \quad (3.3)$$

Porém, como a definição dos custos foi adaptada, o ROI Financeiro também deve ser adaptado considerando as alterações feitas, passando a ser escrito conforme a Eq. (3.4).

$$ROI\ Financeiro = [ROI\ de\ testes\ manuais] + [ROI\ de\ testes\ automatizados] \quad (3.4)$$

3.6 Execução de Testes

Para aplicação da metodologia, os testes foram realizados em um *tablet* do parceiro do CQTS e tanto os testes manuais quanto os automatizados consideraram o mesmo caderno de testes e os mesmos casos de teste. Foram considerados nos testes manuais apenas os testes que foram automatizados, para que a comparação pudesse ser o mais realista possível.

- Resultados dos testes manuais

Pelo fato dos dados dos testes do CQTS serem sigilosos, as tabelas com os resultados presentes nesse trabalho, apresentarão apenas o *Test Case* e o resultado do teste, que pode ser: OK, no caso de sucesso; e, NOK, no caso de defeito.

A planilha de testes que foi repassada por um dos membros da equipe do CQTS, veio com o resultado dos testes executados manualmente e para que não houvessem dúvidas, os procedimentos de teste foram realizados novamente. Os resultados encontrados, podem ser vistos na Tab. (9).

Nos testes manuais, dos 25 casos de teste executados, 18 obtiveram sucesso e 7 apresentaram defeitos.

- Registro de esforço da execução dos testes manuais

Os membros do CQTS, responsáveis pela execução do caderno de testes em questão, registraram o tempo em minutos gasto para a execução de todo o caderno de testes e o resultado encontrado pode ser visualizado na Fig. (11).

O procedimento adotado para registro do tempo gasto para execução apenas dos casos de teste que eram automatizado foi o mesmo e o resultado pode ser observado na Fig. (12).

Tabela 9 – Resultado da execução dos testes manuais

Número do caso de teste	Test Case	Resultado
01	CT10	NOK
02	CT60	OK
03	CT70	OK
04	CT90	OK
05	CT97	OK
06	CT98	OK
07	CT100	OK
08	CT101	OK
09	CT102	OK
10	CT104	OK
11	CT105	OK
12	CT106	OK
13	CT109	OK
14	CT112	OK
15	CT113	NOK
16	CT116	OK
17	CT117	OK
28	CT121	OK
19	CT124	OK
20	CT125	OK
21	CT126	NOK
22	CT129	NOK
23	CT158	NOK
24	CT162	NOK
25	CT166	NOK

Status	Total	Tablet	Total de CT	CT testados	% de conclusão	Minutos/CT	Esforço Total(min)
OK	92	10"	217	156	71,89	3,46	540
NOK	8						
Not Tested	46						
Not Apply	10						

Figura 11 – Modelo de registro do tempo gasto para execução dos testes manuais

Status	Total	Tablet	Total de CT	CT testados	% de conclusão	Minutos/CT	Esforço Total(min)
OK	18	10"	217	25	11,52	4,04	101
NOK	7						

Figura 12 – Registro do tempo gasto para execução dos testes manuais

O tempo adotado para calcular o esforço de execução dos testes manuais será o de 4 minutos por caso de teste, que é a média encontrada na execução dos 25 casos de teste em questão.

- Resultados dos testes automatizados

No caso dos testes automatizados, os *scripts* foram repassados para que houvesse a execução dos testes para a obtenção dos resultados e registro do tempo de execução. Assim como na tabela que apresentou os resultados dos testes manuais, a tabela (10) apresenta apenas o *Test Case* e o resultado do teste.

Os testes em questão foram executados tanto individualmente, ou seja, caso de teste a caso de teste, quanto em conjunto, ou seja, a suíte de teste.

Tabela 10 – Resultado da execução dos testes automatizados

Número do caso de teste	<i>Test Case</i>	Resultado
01	CT10	NOK
02	CT60	NOK
03	CT70	OK
04	CT90	OK
05	CT97	OK
06	CT98	NOK
07	CT100	OK
08	CT101	NOK
09	CT102	NOK
10	CT104	NOK
11	CT105	NOK
12	CT106	NOK
13	CT109	NOK
14	CT112	OK
15	CT113	NOK
16	CT116	NOK
17	CT117	NOK
28	CT121	NOK
19	CT124	NOK
20	CT125	OK
21	CT126	NOK
22	CT129	NOK
23	CT158	NOK
24	CT162	NOK
25	CT166	NOK

Na execução dos casos de teste automatizados, dos 25 casos de teste, apenas 6 obtiveram sucesso, o restante, 19, apresentaram defeito.

- Registro de esforço da execução dos testes automatizados

Ao finalizar-se a execução dos testes automatizados, no arquivo gerado na pasta *reports* é apresentado o tempo de execução para um dos casos de teste e o tempo de

execução da suíte completa. O tempo final encontrado para a execução da suíte dos 25 casos de teste pode ser observado na Fig.(13).

Status	Total	Tablet	Total de CT	CT testados	% de conclusão	Minutos/CT	Esforço Total(min)
OK	6	10"	25	25	100,00	1,44	36,08
NOK	19						

Figura 13 – Registro do tempo gasto para execução dos testes automatizados

O tempo adotado para calcular o esforço de execução dos testes automatizados será o tempo aproximado de 01 minuto e meio por caso de teste, que é a média encontrada na execução dos 25 casos de teste em questão.

- Esforço para a construção de *scripts*

Para que seja possível calcular o esforço despendido na execução de testes automatizados, sabe-se que o esforço de construção de *scripts* deve ser considerado. Para a obtenção desse dado, o Redmine, a ferramenta de gestão de testes do CQTS, foi consultado e os registros lá encontrados permitiram que o tempo gasto, em média, para a construção de um *script* de teste fosse calculado. Os resultados encontrados no Redmine forneceram os dados apresentados na Fig. (14).

Total de CT	Minutos/CT	Esforço Total(min)
96	91,44	8778

Figura 14 – Cálculo da média de esforço para construção de *scripts* com dados do Redmine

Dessa forma, encontrou-se uma média de, aproximadamente, 91 minutos, ou seja, 1 hora e 31 minutos para a construção de cada *script*, e esse será o valor considerado para o cálculo do ROI.

3.7 Aplicação do modelo para cálculo de ROI

A metodologia de Emam (2005) diz que deve-se definir a *baseline* que será considerada no cálculo do ROI, e Pressman (2006) define *baseline* como sendo "[...] um marco de referência no desenvolvimento de um software, que é caracterizado pela entrega de um ou mais itens de configuração", ou seja, determinada versão de um projeto. No caso da presente aplicação de metodologia, será considerada a amostra de um *tablet* de 10"do parceiro do CQTS com o Android 4.0.4. Essa será a *baseline* considerada.

Os passos 1 e 2 - Identificação das atividades que serão observadas do ciclo de vida de detecção de defeitos do CQTS e a definição das medidas que serão utilizadas para

viabilizar a aplicação da metodologia no contexto proposto - que sucedem a definição da *baseline*, que foram definidos no referencial teórico e basicamente consistem no mapeamento das atividades de detecção de defeitos, também já foram realizados e podem ser visualizados na seção 3.2 e, mais detalhadamente, no Anexo A.

- Cálculo da eficácia

Após realizadas essas etapas, dá-se início ao cálculo da eficácia, que é definido pela Eq. (3.5).

$$|\rho_f| = \frac{|\lambda_f|}{|\alpha_f|} \quad (3.5)$$

Onde, $|\lambda_f|$ representa o número de defeitos encontrados durante a atividade de detecção de defeitos; e $|\alpha_f|$ representa o número de defeitos encontrados antes da atividade. Para efeito de comparação, testes manuais serão considerados como sendo atividades realizadas antes da detecção de defeitos e testes automatizados serão considerados como atividades que acontecem durante a detecção de defeitos. Isso vai permitir encontrar a eficácia de um em relação ao outro.

Segundo o resultado da execução dos testes, tem-se que na execução de testes manuais foram encontrados 7 defeitos, ao passo que na execução de testes automatizados foram encontrados 19, logo:

$$|\rho_f| = \frac{19}{7} \approx 2,71$$

Uma eficácia de 2,71 significa dizer que a execução de testes automatizados foi capaz de encontrar quase o triplo de defeitos que a execução de testes manuais encontrou.

- Cálculo do esforço de detecção de defeito

A segunda medida importante é o esforço empregado para encontrar um defeito. Conforme foi definido na seção 3.3, o esforço total é a soma do esforço para construir um *script* com o esforço para executá-lo. Dessa forma, tem-se:

$$E_f = E_{1,f} + E_{2,f}$$

$$E_f = 91 + 1,5 = 92,5 \text{ min/CT}$$

A equação (3.6) traz o esforço por defeito.

$$\varepsilon_f = \frac{E_f}{|\lambda_f|} \quad (3.6)$$

Aplicando os resultados na fórmula, tem-se:

$$\varepsilon_f = \frac{92,5}{19} \approx 4,87 \text{ min/defeito}$$

Emam (2005) considera ainda fases dentro de um processo de detecção de defeitos, que nada mais são do que atividades de detecção de defeitos, e de cada uma dessas fases, pode-se derivar a eficácia e o esforço em diferentes cenários. O autor afirma ainda que na transição de uma fase para a outra é possível que alguns defeitos escapem entre as fases e é possível calcular o valor desse escape ($|\chi_f|$), porém, na aplicação em questão existe apenas uma fase, pois não se considera a correção de defeitos após o relatório enviado ao parceiro do CQTS e uma nova amostra de teste, por isso, o valor do escape vai ser sempre igual a 1. Dessa forma, o esforço de detecção de defeitos é dado pela Eq. (3.7).

$$\text{Esforço de Detecção de Defeito} = \sum_{t=1}^n \varepsilon_f \times \rho_f \times |\alpha_f| \times |\chi_f| \quad (3.7)$$

E substituindo os valores encontrados na equação (3.7), tem-se:

$$\text{Esforço de Detecção de Defeito} = 4,87 \times 2,71 \times 7 \times 1 \approx 92,4 \text{ min/defeito}$$

- Cálculo da economia de esforço

Após calculado o esforço de detecção de defeito, deve-se calcular a economia de esforço através da Eq. (3.8).

$$\text{Economia de esforço} = \sum_{t=1}^n \varepsilon_f \times \rho_f \times |\lambda_f| \times (1 - \beta) \times |\chi_f| \quad (3.8)$$

Onde, os termos $(1 - \beta) \times |\chi_f|$ devem nos dar a diferença de custo entre duas fases, porém, na aplicação está sendo considerada apenas uma fase, logo, o valor atribuído ao produto desses termos será igual a 1. Dessa forma, tem-se:

$$\text{Economia de esforço} = 4,87 \times 2,71 \times 19 \times 1 \approx 250,76 \text{ minutos}$$

- Cálculo da taxa de redução de defeito

Após o cálculo da taxa de economia de esforço, sucede-se o cálculo da taxa de redução de defeito que é dado pela Eq. (3.9).

$$\text{Redução de defeito} = |\alpha| \times \rho_f \times |\chi_f| \quad (3.9)$$

A equação que calcula a taxa de redução de defeitos, também considera o escape entre as fases, e conforme justificado anteriormente, pelo fato de a presente aplicação da metodologia possuir apenas uma fase, esse valor é igual a 1. Logo, substituindo os valores na equação (3.9) tem-se:

$$\text{Redução de defeito} = 7 \times 2,71 \times 1 \approx 19 \text{ defeitos}$$

O detalhamento da metodologia de Emam (2005) considera, ainda, um modelo de economia de custos voltado para o cliente, mas o foco da presente aplicação não está voltado ao repasse do projeto ao cliente, e sim nos resultados que consideram o investimento em automação de testes vantajoso ou não para o CQTS.

3.7.1 O modelo financeiro de ROI

Conforme definições anteriores, o modelo financeiro de ROI para a aplicação em questão é representado pela equação (3.4). Para calcular o ROI de testes manuais, a equação (3.10) deve ser considerada.

$$\text{ROI testes manuais} = \rho_f \cdot \frac{(\sum_{t=1}^n \varepsilon_f \times \rho_f \times (1 - \beta) \times \chi_f) - \varepsilon_f}{\varepsilon_f \times \rho_f \times \chi_f} \times \frac{1}{CM_M} \quad (3.10)$$

Apenas o custo de projeto será considerado, uma vez que os testes geralmente são realizados em computadores pessoais, o que impede de adotar um valor de infraestrutura para o laboratório. O custo de projeto é encontrado considerando o valor recebido pela equipe do CQTS mensalmente (R\$ 700,00), a quantidade de horas por eles trabalhadas (16 horas semanais, considerando que um mês útil possui em média 22 dias) e a quantidade de testadores que participaram dos testes do referido caderno de testes, testando-os manualmente. Aplicando a fórmula apresentada na equação (3.1), encontrou-se o valor aproximado de R\$ 6,76. Substituindo os valores encontrados na equação (3.10), tem-se:

$$\text{ROI testes manuais} = 2,71 \cdot \frac{(4,87 \times 2,71 \times 1) - 4,87}{4,87 \times 2,71 \times 1} \times \frac{1}{6,76} \approx 0,25$$

Já o ROI de testes automatizados, considera a Eq. (3.11).

$$\text{ROI testes automatizados} = \rho_f \cdot \frac{(\sum_{t=1}^n \varepsilon_f \times \rho_f \times (1 - \beta) \times \chi_f) - \varepsilon_f}{\varepsilon_f \times \rho_f \times \chi_f} \times \frac{1}{CA_A} \quad (3.11)$$

Para o ROI de testes automatizados, considerando os valores levantados e as condições idênticas a dos testes manuais, encontrou-se um custo de projeto de R\$ 5,96. E, substituindo na equação (3.11), tem-se:

$$\text{ROI testes automatizados} = 2,71 \cdot \frac{(4,87 \times 2,71 \times 1) - 4,87}{4,87 \times 2,71 \times 1} \times \frac{1}{5,96} \approx 0,29$$

A metodologia de Emam (2005) sugere ainda um cálculo de ROI para redução de prazo de desenvolvimento, chamado *Schedule Reduction*, porém, de acordo com o autor, "[...] o ROI de cronograma se aplica apenas para a parte de pré-lançamento de um projeto de software, podendo-se combinar com o modelo *SCEDRED*". Conforme explicitado no referencial teórico, o cálculo de redução de prazo leva em consideração fatores associados à programação e ao desenvolvimento de sistemas, o que não é o foco do contexto ao qual a metodologia está sendo aplicada. Poderia se levar em consideração o esforço para construção dos cadernos de teste, porém tem-se que os cadernos de testes já vem prontos do parceiro do CQTS. Sendo assim, o chamado *Schedule Reduction* não se aplica ao contexto em questão.

3.8 Interpretação dos Resultados

Levando em consideração os resultados encontrados após a aplicação da metodologia em dois tipos de testes do CQTS, aplicados ao mesmo caderno de testes, é possível observar que os números encontrados mostram que a eficácia dos testes automatizados em relação aos testes manuais ($|\rho_f| = 2,71$), no caderno de testes em questão, é significativa, permitindo que seja possível encontrar quase 3 vezes mais defeitos com os testes automatizados do que com os testes manuais, nesse caso.

No que diz respeito ao ROI encontrado nos dois casos, é possível visualizar que ao serem analisados separadamente, os testes automatizados obtiveram um retorno sobre investimento discretamente maior do que os testes automatizados (0,29 a 0,25). Dessa forma, é possível concluir que o ROI encontrado no caso em que foi aplicado, indica que o investimento em testes automatizados no contexto do CQTS pode ser um bom investimento, pois apresenta vantagens em relação aos testes manuais. Porém, como dito anteriormente a respeito de automação de testes, é imprescindível que ela esteja associada à uma consolidada rotina de testes manuais, uma vez que se complementam.

4 Considerações Finais

Investir em qualidade de software pode ser um diferencial de empresas que querem adentrar no competitivo mercado de desenvolvimento de software. Segundo [Estácio e Valente \(2010\)](#), qualidade do produto de software é importante “[...] por ser um fator que influencia diretamente na competitividade de mercado e prima, sobretudo, em satisfazer as necessidades e requisitos do cliente, da melhor forma possível”. E como é sabido, a qualidade de software visa garantir que o produto de software esteja em conformidade com as especificações explícitas e as necessidades implícitas.

A análise de ROI surge baseada na necessidade de descobrir o quão rentável pode ser esse investimento, de saber quais os benefícios a curto e longo prazo que podem vir de tal investimento e de identificar quais fases do projeto custam mais.

No caso do CQTS, a análise de ROI foi realizada considerando apenas o âmbito da etapa de testes, porém ainda assim, foi possível perceber que investir em automação de testes, no caderno de testes analisado, foi rentável para o CQTS, ou seja, o objetivo geral foi alcançado, pois foi possível perceber pequenos ganhos e avaliá-los.

Lidar com um tema novo e com uma realidade distinta da realidade a que a metodologia utilizada se propõe a avaliar, foi bastante desafiador, uma vez que permitiu testar a habilidade de se adaptar a situações diferentes às pertencentes a uma zona de conforto.

Ao fim da aplicação da metodologia foi possível perceber que os resultados esperados, previamente estabelecidos, foram alcançados e os questionamentos levantados pelos mesmos foram respondidos. Considerando o contexto analisado, que coloca os testes automatizados frente aos testes manuais, tem-se que investir em melhoria da qualidade, automatizando os testes, nesse caso, é mais vantajoso ao laboratório.

4.1 Sugestões de melhoria para o CQTS

O período dedicado à coleta de dados no laboratório do CQTS permitiu que fossem observados pontos importantes que resultaram em algumas sugestões de melhoria, descritas a seguir:

- Deveria ser estabelecida uma política de conscientização da importância da utilização da ferramenta de gestão de projetos no laboratório. Tal ação é útil, principal-

mente, para os gestores do laboratório, para que possam ter controle e saber como está o andamento da realização dos testes.

- Deveria ser elaborado um padrão de planilha para registro dos resultados, pois não existe. Foi observado que o resultado de cada caderno de testes é documentado e relatado de uma forma, sem que exista um padrão. Foram identificados pelo menos três tipos de planilha de registro de resultado de testes diferentes, o que não uniformiza o conhecimento do funcionamento dessas planilhas entre os bolsistas do CQTS. De certa forma, cada um entende de um jeito como os resultados devem ser documentados.
- Talvez a realização de palestras de alinhamento de conhecimento dos processos do CQTS poderia ser útil, uma vez que foi percebido que nem todos os bolsistas estavam a par do funcionamento dos processos. Isso auxiliaria, inclusive, no entendimento do papel de cada um dentro do laboratório.

4.2 Sugestões de trabalhos futuros

Sugere-se, baseada na metodologia utilizada, a criação de uma metodologia própria para a avaliação de ROI em contexto de fábrica de teste de software, considerando que a mesma não faça parte do processo de desenvolvimento de software. Isso possibilitaria que empresas que terceirizam esse serviço saibam em que melhorias investir e onde focar o investimento, a fim de ampliar seus lucros.

Sugere-se, ainda, que a metodologia utilizada seja aplicada a outros cadernos de testes afim de que se tenha um resultado mais consistente.

Referências

- BARTIÉ, A. *Garantia da Qualidade de Software: Adquirindo Maturidade Organizacional*. São Paulo, Brasil: Campus, 2004. Citado 6 vezes nas páginas [21](#), [26](#), [27](#), [28](#), [29](#) e [30](#).
- BOEHM, B.; BASILI, V. Software defect reduction top 10 list. 2007. IEEE Software, n. 34, p. 135–137, 2007. Citado na página [32](#).
- CAETANO, C. Engenharia de software magazine. 2008. DevMedia, Rio de Janeiro, n. 4, p. 60–66, 2008. Citado na página [33](#).
- DEMARCO, T. *Controlling software projects : management, measurement & estimation*. New York: Yourdon Press, 1982. Citado na página [29](#).
- DIAZ, M.; SLIGO, J. How software process improvement helped motorola. 1997. IEEE Software, p. 75–81, 1997. Citado na página [38](#).
- EMAM, K. E. *The ROI from Software Quality*. Florida: Auerbach Publications, 2005. Citado 17 vezes nas páginas [17](#), [22](#), [36](#), [39](#), [40](#), [41](#), [43](#), [47](#), [50](#), [53](#), [54](#), [55](#), [56](#), [60](#), [62](#), [63](#) e [64](#).
- ESTÁCIO, B.; VALENTE, C. Engenharia de software magazine. 2010. DevMedia, Rio de Janeiro, n. 31, p. 14–18, 2010. Citado 2 vezes nas páginas [29](#) e [67](#).
- GUERRA, A. C.; COLOMBO, R. M. T. *Tecnologia da Informação: Qualidade de Produto de Software*. Brasília: PBQP Software, 2009. Citado 2 vezes nas páginas [25](#) e [26](#).
- ISO/IEC. *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. Geneva, 2010. Citado 4 vezes nas páginas [17](#), [26](#), [27](#) e [28](#).
- JAYACHANDRAN, N. *Understanding ROI Metrics for Software Test Automation*. Florida: University of South Florida, 2005. Citado 2 vezes nas páginas [54](#) e [55](#).
- KALLNOWSKI, M.; SPINOLA, R. O. Engenharia de software magazine. 2007. DevMedia, Rio de Janeiro, n. 1, p. 68–74, 2007. Citado 3 vezes nas páginas [15](#), [31](#) e [32](#).
- KANER, C.; FALK, J.; NGUYỄN, H. *Testing computer software*. Michigan: Van Nostrand Reinhold, 1993. Citado na página [33](#).
- KUSUMOTO, S. *Quantitative Evaluation of Software Reviews and Testing Processes*. Osaka: Osaka University, 1993. Citado na página [41](#).
- LAGES, D. S. Engenharia de software magazine. 2010. DevMedia, Rio de Janeiro, n. 29, p. 20–25, 2010. Citado na página [33](#).
- LEAL, J. A usabilidade e o roi(return of investment). 2002. Saulo, 2002. Citado na página [21](#).

- NBR ISO/IEC. *NBR ISO/IEC 14598-1 - Tecnologia da Informação - Avaliação de produto de software Parte 1: Visão Geral*. Rio de Janeiro, 2001. Citado na página 26.
- PFLEEGER, S. L. *Engenharia de Software: Teoria e Prática*. São Paulo: Prentice Hall, 2004. Citado 2 vezes nas páginas 34 e 35.
- PRESSMAN, R. *Engenharia de Software*. Porto Alegre: Bookman, 2006. Citado 4 vezes nas páginas 29, 30, 31 e 60.
- RICO, D. *ROI of Software Process Improvement: Metrics for Project Managers and Software Engineers*. Florida: J. Ross Publishing, Inc., 2004. Citado 2 vezes nas páginas 21 e 38.
- SIKKA, V. *Maximizing ROI on Software Development*. Florida: Auerbach Publications, 2004. Citado na página 37.
- SOLINGEN, R. Measuring the roi of software process improvement. 2004. IEEE Software, p. 32–38, 2004. Citado 3 vezes nas páginas 35, 37 e 38.
- SOLINGEN, R. V.; BERGHOUT, E. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. Londres: McGraw-Hill, 1999. Nenhuma citação no texto.
- SOMMERVILLE, I. *Engenharia de Software*. São Paulo: Addison-Wesley, 2007. Citado 7 vezes nas páginas 15, 29, 30, 31, 32, 33 e 34.

ANEXO A – Primeiro Anexo

A.1 Descrição dos processos do CQTS

O Anexo I traz o detalhamento dos fluxos de processo mapeados no CQTS, o processo de "Gerenciamento de Demanda" e o processo denominado "Processo de teste"

A.1.1 Gerenciamento de Demanda

O fluxo de "Gerenciamento de Demanda" pode ser observado na Figura (15) e obedece à seguinte descrição.

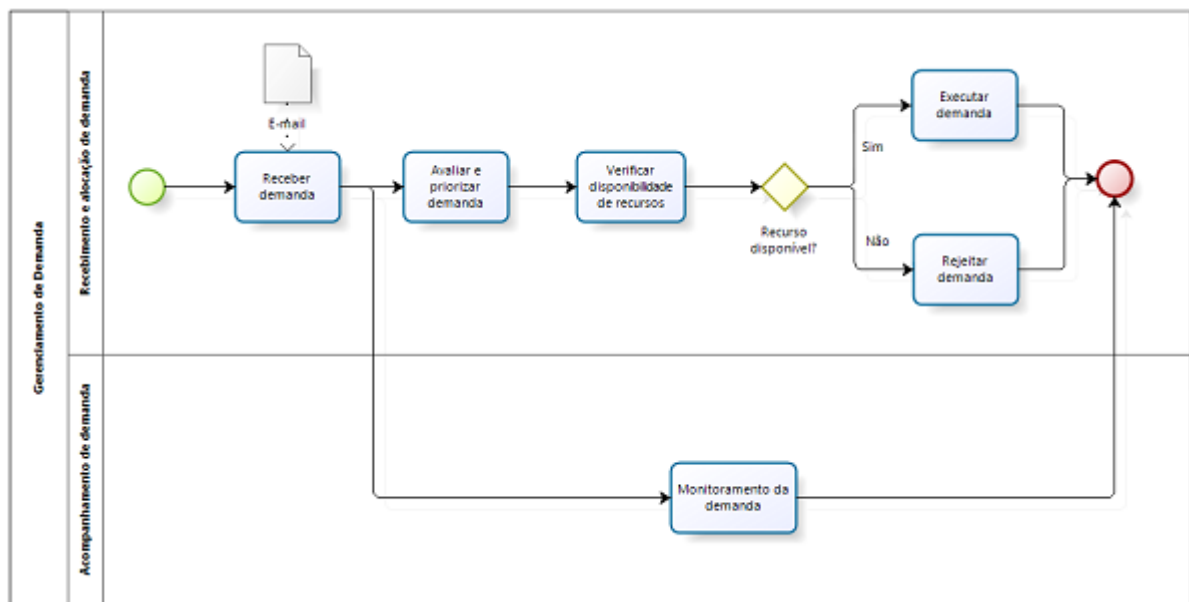


Figura 15 – Processo de Gerenciamento de Demanda do CQTS.

- Tarefas

Receber demanda – Na tarefa “Receber demanda”, as demandas são repassadas por e-mail por responsáveis do parceiro do CQTS para a equipe do Centro de Qualidade e Teste de Software (CQTS).

Avaliar e priorizar demanda – Após serem recebidas pelo CQTS, as demandas são avaliadas pelo responsável do laboratório e tem sua prioridade discutida entre

o responsável do laboratório e os responsáveis do parceiro do CQTS e, a partir daí, a prioridade das demandas é decidida.

Verificar disponibilidade de recursos – Após a priorização das demandas, a disponibilidade dos recursos que serão utilizados para sua execução – sejam eles físicos ou humanos – são verificados.

Executar demanda – No caso de haver disponibilidade de recursos, a demanda é executada.

Rejeitar demanda – No caso de haver impossibilidade de execução da demanda, pelo fato de não haver disponibilidade de recursos, a demanda é rejeitada.

Monitoramento da demanda – A partir do momento que é recebida, a demanda é monitorada até o momento da sua finalização, através da ferramenta de gestão (Redmine) e de reuniões que são realizadas semanalmente afim de se obter o feedback do andamento das demandas.

- Condicionais

Recurso disponível? – Tal condicional verifica a disponibilidade dos recursos físicos e humanos do laboratório para a execução de uma demanda.

A.1.2 Processo de Teste

O fluxo de "Processo de Teste" pode ser observado na Figura (16) e obedece à seguinte descrição:

A.1.2.1 Planejamento da Demanda

- Tarefas

Definir Estratégia de Testes – Esta tarefa se destina à definição da estratégia a ser adotada para cada caso de teste – Execução, Criação ou Automatização de Testes. Cada estratégia desemboca em um novo fluxo de execução de Processo de Teste.

- Condicionais

Caso de teste automatizável? – Tal condicional define se o caso de teste em questão é ou não automatizável, o que irá ajudar a definir a estratégia de testes a ser adotada. **Execução de Testes?** – Define qual a estratégia de testes a ser adotada. Caso não seja “Execução de Testes”, será “Construção de Testes”.

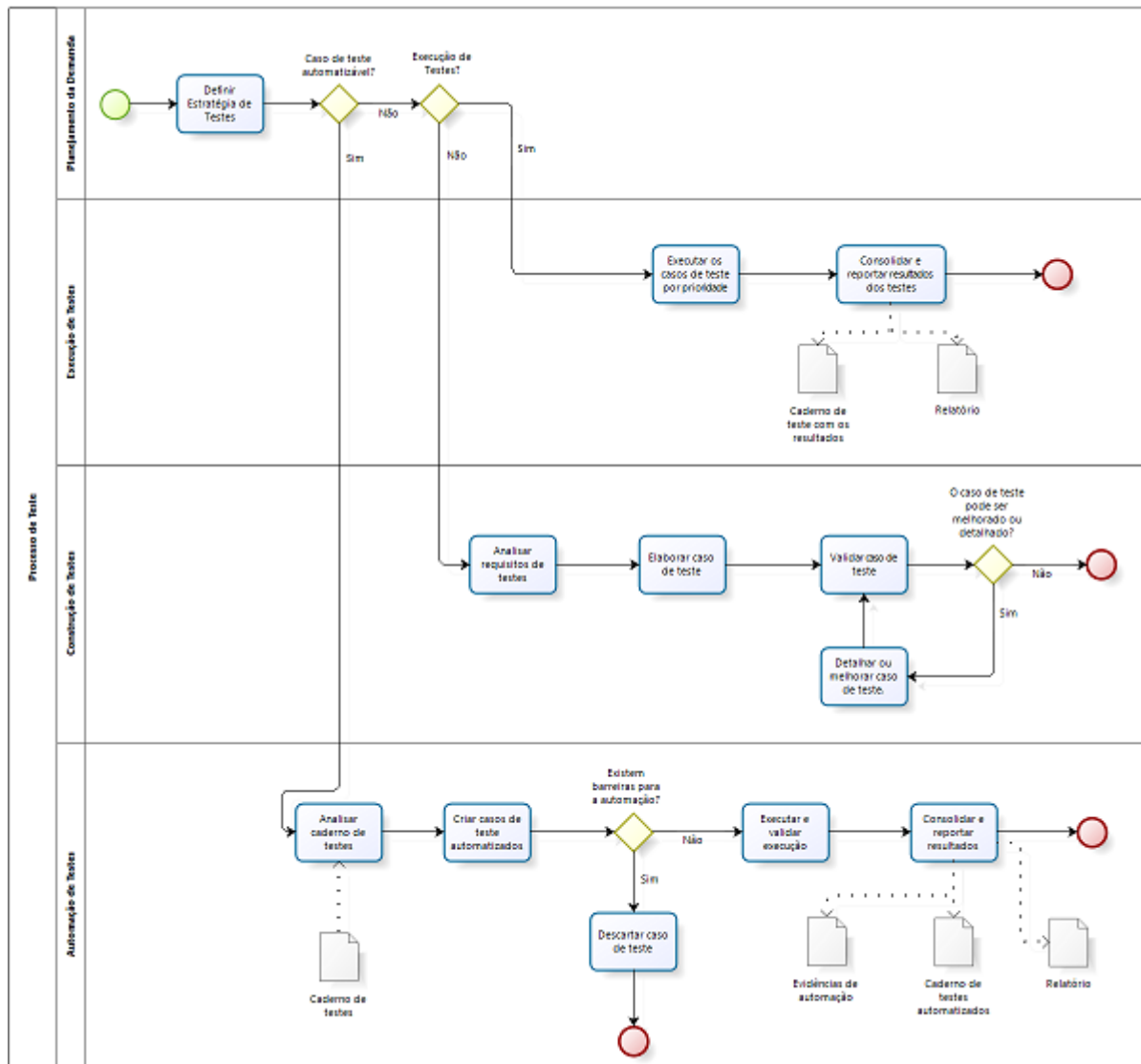


Figura 16 – Processo de Testes do CQTS.

A.1.2.2 Execução de Testes

- Tarefas

Executar os casos de teste por prioridade – Os casos de teste serão executados conforme prioridade definida no Fluxo de Gerenciamento de Demanda.

Consolidar e reportar resultados dos testes – Ao fim da execução dos testes os resultados serão consolidados e serão gerados dois artefatos: o caderno de testes, que trará os resultados da execução dos testes, e o relatório de execução dos testes.

A.1.2.3 Construção de Testes

- Tarefas

Analisar os requisitos de teste – Nesta tarefa os requisitos de teste, ou seja, o que é esperado como resultado do teste e o que deve ser testado, são analisados.

Elaborar caso de teste – Os casos de teste são elaborados com base nos requisitos analisados.

Validar caso de teste – Após a elaboração, os casos de teste devem passar por validação para verificar se há a necessidade de melhorá-los.

Detalhar ou melhorar caso de teste – Caso a validação indique que há necessidade, nessa etapa os casos de teste são detalhados ou melhorados de acordo com o que for identificado na tarefa anterior.

- Condicionais

O caso de teste pode ser melhorado ou detalhado? – A condicional define a necessidade de detalhamento ou melhoramento do caso de teste construído.

A.1.2.4 Automatização de Testes

- Tarefas

Analisar caderno de testes – É realizada uma análise do caderno de testes, para que seja possível perceber quais casos de teste são passíveis de automação e quais não são.

Criar casos de testes automatizados – Os casos de testes são automatizados conforme definição da etapa de planejamento, onde há a definição se são automatizáveis ou não.

Descartar caso de teste – Caso exista algo que impeça a automação de ser realizada, o caso de teste deve ser descartado.

Executar e validar execução – Os casos de teste automatizados são executados e tem sua execução validada.

Consolidar e reportar resultados – Ao fim da automação dos testes os resultados serão consolidados e serão gerados três artefatos: as evidências de automação, os cadernos de testes automatizados, que trará os resultados da execução dos testes, e o relatório de execução dos testes.

- Condicionais

Existem barreiras para a automação? – Essa condicional verifica se existe algo que impeça a automação de testes de ser realizada.